Coding Neural Networks and training them for Face Recognition

Final Report

Subject Module Project in Computer Science
Roskilde University, Spring 2019
27 May 2019

Abdul Halim bin Abdul Rahman
Rasmus Grevinge Jensen
Vit Zemanek

Supervised by
Bjarne Poulsen

Subject Module Project in Computer Science

Department of Human and Technology

Roskilde University, Spring 2019

Research Title

Coding Neural Networks and training them for Face Recognition

Research Question

What are the underlying mathematical concepts behind neural networks and how are they implemented for face recognition?

Authored by

Abdul Halim **bin Abdul Rahman**
63870 @ NIB 2017
ahbar@ruc.dk

Rasmus Grevinge **Jensen**
63753 @ HUMTEK 2017
ragrje@ruc.dk

Vit **Zemanek**
62643 @ NIB 2017
vit@ruc.dk

Supervised by

Bjarne **Poulsen**
External Supervisor
bjarnep@ruc.dk

Every facial image that appear in this report is used with explicit consent from respective volunteer. Unless otherwise stated; charts, diagrams, figures, graphs and/or any other graphics composite were made using free web-tool available at *www.draw.io* or plotted using Python's *Matplotlib*.

ABSTRACT

There are a number of machine learning libraries available on the internet with high abstraction, hence the mathematical processes behind machine learning and neural network in particular are pushed deeper into the background that software developers do not need to concern themselves with it. Hence the motivation of this research to shed light to some of these mathematical concepts and to show how they can be implemented by coding a neural network without the use of machine learning libraries. Neural networks can be used for many different purposes, but face recognition was chosen due to it having some interesting mathematical intricacies on its own and because it provided an opportunity to build a dataset from the ground up. Several experiments were conducted, the results of which are discussed; with achievement of 95% accuracy of balanced dataset. Ideas for future research are provided at the end of the report.

KEYWORDS

Artificial Neural Network, Backpropagation, Cross Entropy, Face Recognition, Forward Propagation, Machine Learning, Momentum Gradient Descent, LBPH, Local Binary Patterns, Logistic Regression, Stochastic Gradient Descent, Sigmoid, Softmax.

PERSONAL ACKNOWLEDGEMENTS

“I would like to thank our supervisor Bjarne Poulsen who has been supportive and for his productive input that helped this project to be what it is today; together with the opponent group, Max, Frederik and Poul for their valuable inputs and comments. Also, I would like to record a very big thank you to the volunteers who had helped with facial images for the experiment: students from Roskilde University, classmates from Taastrup Sprogskole; for being so ever helpful to spending their valuable time for photography sessions, being so sporting and supportive throughout the research period. Hoping to receive similar support for future project. Gratitude goes to Henning Christiansen for valuable materials and also for allowing me to follow his teaching for master’s student’s *Artificial Intelligence* class; and the admin of INM’s *High Performance Computer*, Heine Larsen, I would be forever grateful for the access to the server. Not to forget, Ole Torp Lassen for his forever useful mantra, *“don't leave a stone unturned.”* Last but not least, to everyone who has been there to listen when I moaned, to read the early drafts and later coming back with valuable feedback as well those who have been curious and tossing questions into my direction that keep on motivating me to think over a certain matter and look-up for correct answers.”

Halim

Glossary

ANN	Artificial Neural Network
Back propagation	A series of steps consisting of algorithms to find more accurate weights.
CNN	Convolutional Neural Network
Face Detection	The ability to detect what a face is and distinguishing it from other things e.g. distinguishing a face from a bicycle or an arm.
Face Recognition	Being able to recognize/validate the characteristics of a specific face.
Forward propagation	A series of steps consisting of algorithms to transform input data.
GPR	Geometric Pyramid Rule
GPU	Graphics Processing Unit
Histogram	Representation of the distribution of numerical data.
HOG	Histogram of Oriented Gradients
LBP	Local Binary Patterns
LBPH	Local Binary Pattern Histograms
MVC	Model-View-Controller
NN	Neural network
OOP	Object Oriented Programming
UML	Unified Modeling Language

Table of Contents

CHAPTER

①

Research Introduction and Project Planning Overview

This chapter will provide examples of a few cases that have inspired the research question, and a quick introduction into the research domain and relevant methodology, which will be analysed in depth in chapter 2, 3 *and* 4*. In order to complete this research satisfactorily, the visions, requirements, goals and plans will be laid out so that they can be used as a proper guide of reference during the period of this research.*

1.1 MOTIVATION

Neural network is a concept that was introduced in 1943 by Warren McCulloch and Walter Pitts [1][2]. With the advancement of technology in terms of hardware and data availability, it has become widely implemented and these days, its implementation is becoming easier and quicker when appropriate programming libraries are used. However, the use of libraries also means that an individual programmer is no longer required to have an understanding of the underlying mathematical concepts being applied as these libraries have turned them into some abstraction. Neural networks are based on different mathematical fields which leads to intricacies in their foundations that will be explored in chapter 2.

Neural network as a mathematics-based methodology has many different applications, one of them is pattern recognition that can be implemented for face recognition. Face recognition aided by machine learning created a storm in 2018 when Amazon revealed that they had been providing a technology called *Rekognition* to the police department in Orlando and Oregon [3]. In the same year, Dubai International Airport tested a state-of-art face recognition technology [4] *en masse* for counter check-in, baggage-drop, immigration check and speedy boarding process. In the early of 2019, *CaixaBank* installed 20 Automatic Teller Machines (ATM) in Barcelona, to allow transactions to be verified using face recognition instead of typing a

Personal Identification Number (PIN). Based on their survey, 70% of customers said they would be ready for such technology [5].

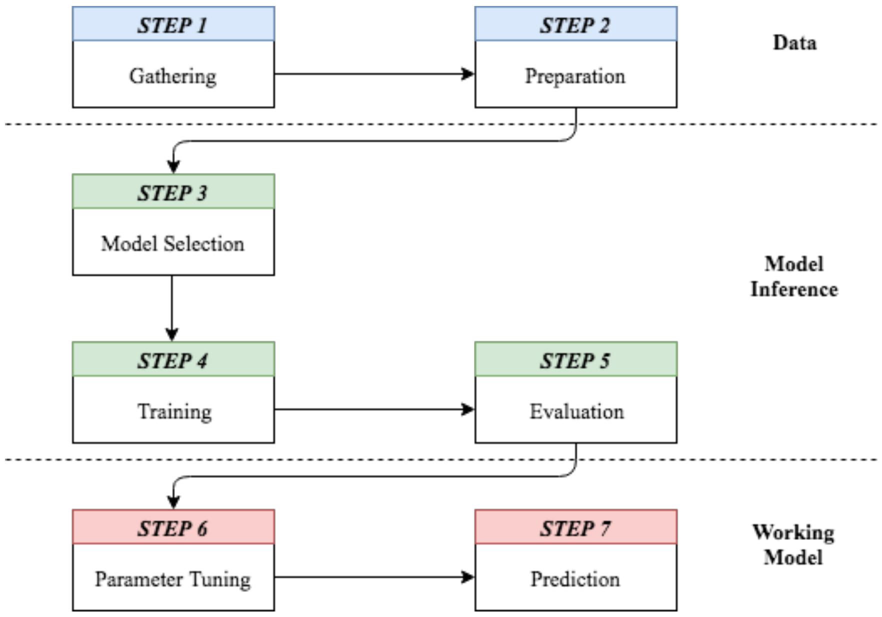


Figure 1 : Seven steps to create a machine learning algorithm [6]. It involves data gathering, model inference and lastly producing a working model.

The strength of neural networks for different tasks comes from training which is also known as the *learning process*. In order to perform well, a neural network must be trained for a specific use with a relevant dataset by exploring the domain of face recognition and collecting a dataset in order to train a neural network. A machine learning project must go through seven steps of a sequence flow which can be seen in Figure 1. The first two steps are related to the dataset, and must be completed before training can take place based on a selected model. The output from the training must be evaluated to measure whether the model is producing the intended answers. Once the training process has been completed, the model can be saved and used for subsequent prediction. In this project, emphasis will be on *model selection*, since this is where important mathematical concepts are being applied and is the core of any given neural network.

The research question to be explored and answered in this report is as follows:

What are the underlying mathematical concepts behind neural networks and how are they implemented for face recognition?

1.2 PROJECT DESCRIPTION

In order to answer the research question, this project must evaluate its goals and visions, problem domains and relevant methodology. Therefore, the scope for this research can be well defined.

1.2.1 Goals and Visions

There are two goals for this project. The first is to do literature review for two problem domains: neural network and face recognition. This will help to understand and explain the mathematics behind neural networks at a conceptual level using calculus, statistics and algebraic formulations; as well as the concept behind face recognition. Each analysis will have a number of possible case studies to be investigated and some may be relevant to be experimented and properly analysed at the end of this report.

The second is to implement two neural network namely Artificial Neural Network (ANN) and Convolutional Neural Network (CNN) based on the understanding of mathematical formulations from the analysis in chapter 2. Some experiment based on proposed case studies will be done using the implemented neural networks.

1.2.2 Neural Networks

Machine learning is a mechanism to enable a computer to learn and progress from *experience* without being explicitly programmed. The goal is to understand the structure of a particular data and fit the data into a particular model that has a proper classification. Neural network is one such implementation methodology for machine learning [7]. It is gaining a lot of traction with interesting breakthroughs [8], one good example is self-driving cars. A neural network is conceptualised through a combination of different mathematical fields as shown in Table 1.

Linear Algebra	**Statistics**	**Calculus**
Forward propagation	*Cross Entropy*	*Backpropagation*
	Probability Distribution and Accuracy	*Optimization*

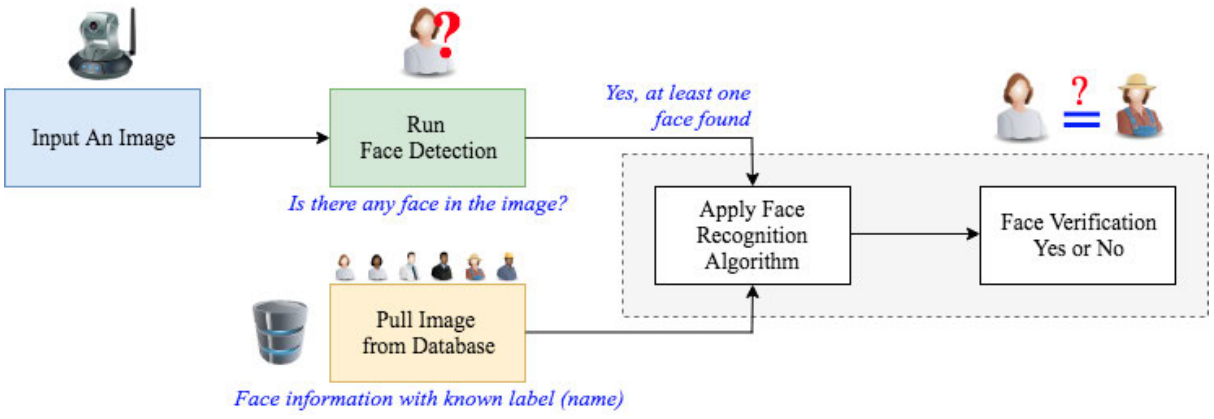
Table 1 : Neural network applies a number of different mathematical fields for its interconnected processes which this research is interested to investigate.

In this research, three different mathematical fields will be explored namely linear algebra, statistics and calculus. It is observed that calculations in neural network can be divided into five main groups of calculations. These are all abstract mathematical concepts that will be explored in depth in chapter 2.

It is becoming relatively easy to implement a neural network thanks to the growing numbers of libraries available freely on the internet and the number of high-level programming languages that are closer to natural language. However, understanding its internal logic from the ground-up will help for further development and improvement to training for a better model as well as when tuning certain parts of a neural network.

1.2.3 Face Detection and Face Recognition

Face detection is a challenge to determine if there is at least one human face in a given image; whereas facial recognition can be defined as a challenge to classify whose face is in an image. For facial recognition to work, a set of images labelled with a name is required [9]. Two facial images can be compared mathematically to see if two images came from the same person, then the numerical difference between them should be very small [10]. An example in daily life for face recognition could be the biometric passports which keeps a facial photo in the chip located somewhere in the passport booklet; when a passport is introduced at an immigration checkpoint, a live photo is taken and then compared to the one in the passport.



Input An Image
Run Face Detection
Is there any face in the image?
Yes, at least one face found
Apply Face Recognition Algorithm
Face Verification Yes or No
Pull Image from Database
Face information with known label (name)

Figure 2 : The flow on top is for face detection which is to determine if there's a face. If the answer is positive, the will it continues with the next process for face verification.

Basic concept for face detection and facial recognition are explained in Figure 2. The input image is passed to algorithms for face detection, when no *face* found, the process stops. If at least one *face* is found, the section that contains the face is extracted. If a particular algorithm for face recognition was applied to images in the database, a similar algorithm must also be applied to the newly extracted data. Lastly, the new image will be compared against each image from the database to produce an identification.

The final process of *Face Recognition* from Figure 2, bounded by the greyed area can be replaced with a neural network (or alternatively, a neural network can use pre-processed data by traditional face recognition algorithms as its input). Unlike other algorithms for face recognition which require only one facial photo, a neural network requires more in order for it to learn their similar features. The greyed-area in the same Figure 2 is the prospect for this research to investigate where the face recognition algorithm will be replaced with a neural network. In this project, it is assumed that images have been processed by face detection algorithm, but the actual process is that the faces will be cropped manually to simulate the face detection algorithm due to time constraints and to give more focus on neural network instead of face detection.

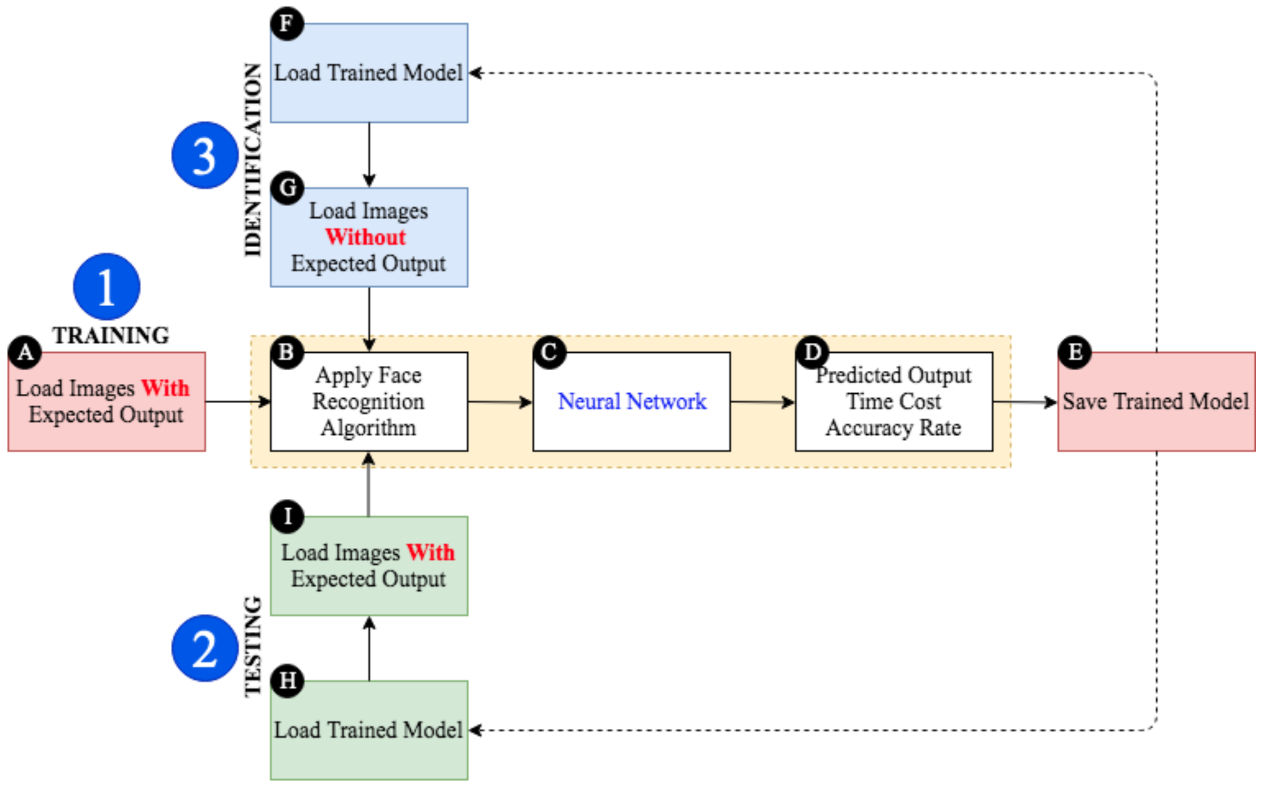

F
Load Trained Model
3
IDENTIFICATION
G
Load Images Without Expected Output
1
TRAINING
A
Load Images With Expected Output
B
Apply Face Recognition Algorithm
C
Neural Network
D
Predicted Output Time Cost Accuracy Rate
E
Save Trained Model
I
Load Images With Expected Output
2
TESTING
H
Load Trained Model

Figure 3 : Flow diagram for face recognition using neural network. Solid arrows show the direction of flow for each numbered process and the dotted arrows are to simulate the passing of the trained model to be used in Process 2 and Process 3.

From Figure 3 above, the first process is *Training*. It is a sequential process and starts from step Ⓐ by loading images and labels. Then the step Ⓑ will apply face recognition algorithm on the loaded images before passing to step Ⓒ for generalization. The output from the neural network in Ⓓ will measure the predicted output as well as the processing time and accuracy rate. If the output gave a good prediction and good accuracy rate, the step Ⓔ can be used to save the trained information so that it can be used for *Testing* or for *Identification*.

The second process is *Testing*. This process can only take place if there is information saved from step Ⓔ where it will be loaded in step Ⓕ. Then in step Ⓖ, it loads images with known labels and passed from step Ⓑ until step Ⓓ where analysis can be done to measure whether the trained model is above certain accuracy rate for future use of face recognition.

The final process is *Identification.* Similarly, in *Testing*, this process will require information saved from step Ⓔ. The step Ⓗ loads the trained data with images but without any label in step Ⓘ. The images and training data are passed to a neural network for data generalization from step Ⓑ until step Ⓓ when it produces an identification for the image.

At the end of the training process, the *trained model* which stores the knowledge about what the neural network has learnt will be saved so that it can be used for testing and identification purposes.

1.3 DESIGN CONSIDERATIONS AND SPECIFICATIONS

This research not only that will be based on theory analysis but also implementing the knowledge. Therefore, the important task needed to be listed down along with the how to build the required specifications in order to bridge between the analysis and a working implementation.

1.3.1 Face Recognition

In order to train a neural network for face recognition it needs to be provided with images for training. The input images must be meaningful for the neural network, otherwise it cannot be trained and applied in a different environment. Algorithms for face recognition will then be applied on images from the dataset.

1.3.2 Neural Network and Dataset

The implementation is intended to be very simple by including only necessary, basic features for neural network to perform well. As the dataset is expected to be rather large and the training will be based on different dimensions of input images, the experiment will be conducted on a High-Performance Computer (HPC) available at the *Department of Science and Environment* located in Building 27 at Roskilde University.

The study of the neural network will be implemented for face recognition. A dataset for this experiment will be built from the ground up by recruiting for volunteers. Volunteers will be asked to sign a consent form permitting his/her facial images to be used for the purpose of this experiment. Cleaning, cropping, resizing and labelling for supervised training will be done manually.

1.3.3 Implementation

Neural network will be programmed using Python based on the Model–View–Controller (MVC) approach. Unified Modelling Language (UML) will also be used for the purpose of describing the planned classes in order to facilitate the actual implementation.

There are no plans to implement a Graphical User Interface (GUI) as the intention of the project is to explore the underlying mathematical mechanisms, not intricacies of GUI. However, should time permit, it may be reconsidered.

1.4 PROJECT PLAN

Time-management and content-management tools used for organization of the project progress are presented here.

1.4.1 Gantt Chart

	Month →	Feb			↔	Mar				Apr				↔	May			↔	Jun			
	Task ↓ Week Number →	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
1	Project Market	→																				
2	Research Proposal (Chapter 1)		→	21																		
3	Data Gathering - Consent Form, Recruiting Volunteers & Collecting Photos			→	→	→	→	→	→	→	→	→	→	→								
4	Dataset - Editing				→	→	→	→	→	→	→	→	→	→	→							
5	Mid-Term Evaluation (Preparation & together with Problem Analysis)				→	→																
6	Mid-Term Evaluation						13															
7	Problem Analysis & Theory (Reading literature & writing)						→	→	→													
8	Implementation (Coding neural network and applying algorithms)								→	→	→	→	→	→								
9	Experiment												→	→	→							
10	Discussion and Conclusion													→	→							
11	Final Draft															15						
12	Internal Deadline																22					
13	Final Submission																	27				
14	Preparation for Final (Presentation, Poster)																	→	→			
15	Preparation for Exam (Questions)																			→		
16	Exam Week																				→	←

↔ The particular week is shared with both previous and following month.
→ is marked for as busy / on going.
Number in a box is the exact date in the particular month.

1.4.2 Outline of Reports

Chapter 2 : Neural Network
It will be explained what neural networks are, their brief history will be introduced. Mathematical concepts that form the core of neural networks are explained. Necessary mathematical formulas are presented.

Chapter 3 : Face Recognition
Basics of face recognition are elucidated. Differences between face recognition, face verification, face detection are pointed out. Algorithms for face detection and face recognition are presented.

Chapter 4 : Dataset
Specifications of the dataset used for the purpose of training the neural network are given. It is elaborated on the relationship between the dataset, face recognition and neural networks.

Chapter 5 : Design and Implementation
Description of software design and brief explanation of design methodology.

Chapter 6 : Case Studies and Experiments
To implement possible case studies from chapter 2, 3 and 4 where the output will be recorded for analysis in Chapter 7.

Chapter 7 : Discussion and Conclusion
Summary from case studies to explain what has been observed from the experiment and the overall summary regarding dataset collection, implementation and experiment will also be discussed. At the end, some discussion regarding possible future works related to topics discussion in the research, where certain angles can be explored.

CHAPTER

②

Theory and Analysis :

Neural Network

The first domain to be analysed is neural network with its brief history and a proper introduction to the architecture which includes the mathematical concept and components that form a working network. After that, some possible case studies related to this topic will be proposed.

2.1 BRIEF HISTORY OF NEURAL NETWORKS

Neural networks is an algorithm that fall under the umbrella of Deep Learning which is a subset of Artificial Intelligence (AI) where AI is defined as *"intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals"* [11] and has improved over the years, thanks to the advancement of software and hardware and is meant to replicate the ability, versatility and creativity of human thinking to a computer [12]. A proper funding was given to Dartmouth Summer Research Project on Artificial Intelligence which was held at Dartmouth College in Hannover, Hampshire in 1956.

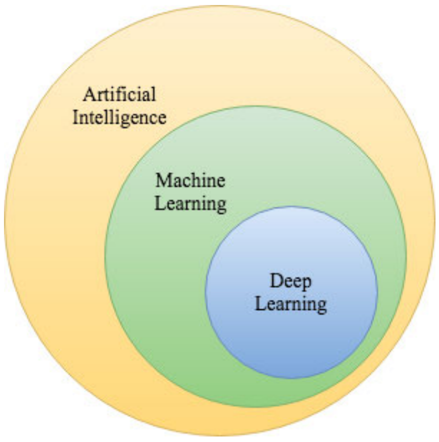


Figure 4 : Deep Learning is a subset to Machine Learning which in turn is a subset to Artificial Intelligence.

A subset in AI is Machine Learning (ML) which has its focus on constructing computer systems that can predict decisions based on previous experience and improve the decision making as when and more trainings are given [13] without exactly knowing that the substance is [14]. The decision process is based on understanding the structure of a given data and fit that data into models that can be utilized later without again looking back at the original data for subsequent decision [15]. Machine learning has been employed in so many tasks in commercial activities, for example classifying spam emails by Google Gmail, enhance-driving car by BMW and image recognition by Amazon. In March 2019, researchers from the University of Nottingham claimed that they have discovered that a machine learning can be used to predict an individual's mortality based on health, dietary, etc. using data from over 500,000 people sampled between 2006 and 2016 [16].

Machine learning problem can be defined as a challenge to improve some measurement of performance, P, when some class of tasks, T, have been executed through some type of training experience, E [13] and can be said is learning if its performance for some particular tasks in T which is being measured by P is improving over experience E [17]. One method to improve E is through Neural Network training which is one of Machine Learning algorithms to deploy Deep Learning [18] which learns by putting weights on certain features to produce a certain output. The first application of Neural Network to solve real-case problem was developed by Bernard Widrow and Marcian Hoff at Stanford in 1959 and named ADALINE that applied neural network for use as an adaptive filter that eliminated the echo on phone lines.

2.2 NEURAL NETWORKS

A neural network is a method of learning by having two inputs, one is a set of features, conventionally denoted as $\mathbf{X}$ and one is a set of correct answers, the expected output, denoted as $\mathbf{Y}$. It then performs a cycle of learning process by making a series of calculation and once a cycle is completed, it calculates the error which is the difference between the output from final activation function against known correct answer. From Figure 6, two principal mathematical components for neural network are denoted as Σ and $\boldsymbol{\sigma}$ for to sum the product of input features $\mathbf{X}$ and its corresponding weight, W; and activation function, respectively. These two are part of forward propagation, at the end of the flow it produces a predicted output, $\hat{\mathbf{Y}}$.

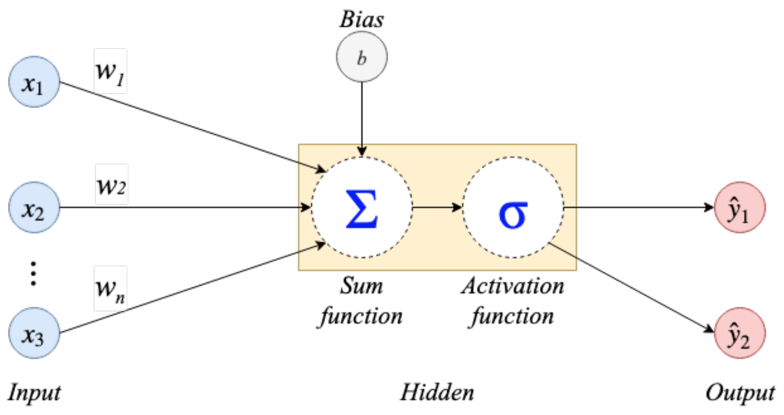


Figure 5 : A basic neural network model with a single neuron to show relations in three layers of input, hidden and output. It consists of a set of inputs, ***x****, weights,* ***w****, and a bias,* ***b****, whose values are passed to a summation function, whose result is then passed to an activation function to yield a set of outputs.*

The concept of a neural network was introduced by Warren McCulloch and Walter Pitts in 1943 [20] and named *'threshold logic'* [21], the figure below is trying to mimic the original design but with two additions namely bias and one additional output. The model contains a number of input, features matrix $X = \{x_1, x_2, , x_n\}$, a set of coefficient vector for weight $W = \{w_1, w_2, , w_n\}$, the sum function which is then squashed using an activation function and a vector for predicted output $\hat{Y} = \{\hat{y}_1, \hat{y}_2\}$ to describe a binary output of 0 and 1. However, in the original diagram, the model lacks bias and the output $\hat{Y}$ was a scalar.

An application from the introduced concept is perceptron. It was proposed by Frank Rosenblatt, a psychologist at Cornell Aeronautical Laboratory, in 1957 as an idea to create a computer that could learn by trial and error [19]. His idea caused a controversy when he affirmed that the perceptron was the beginning for machines that in the future will be able to walk, talk, see, write, and even reproduce itself and be conscious of its existence [19]. His idea, *Mark 1 Perceptron,* was a simple input-output system which was modelled to McCulloch-Pitts' proposal but it was only able to solve linear problem. Rosenblatt's perceptron consists of one or more inputs, a node and only a single output.

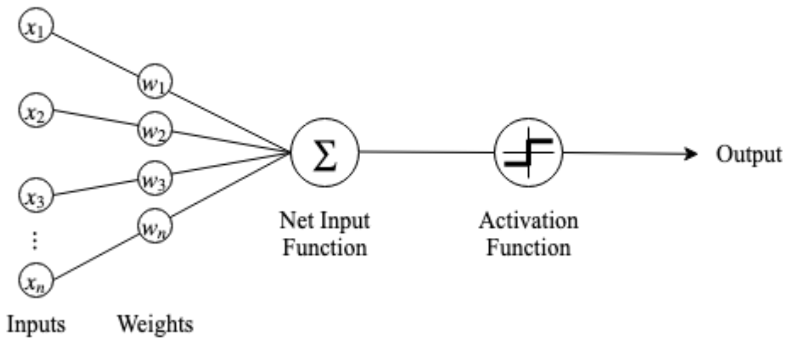


Figure 6 : Schematic of Rosenblatt's Perceptron. It contains a set of input where each one has a specific weight for the sum calculation. The sum value is then passed through a Step function to produce an output.

Rosenblatt's Perceptron has a set of input, $X = [x_1 \quad x_2 \quad x_3 \quad \cdots \quad x_n]$ and each input has a connecting weight, denoted by w_n for each x_n. The first process is calculating the sum of the total input $\sum_{i=1}^{n} x_i \cdot w_i$ and then the value is passed through step-function which produces an output value $\{-1, 1\}$.

The term *neural* suggests that it is intended to replicate the way the human brain learns and makes decisions. A neural network consists of a number of neurons which are grouped into layers. Each neuron in a layer is also known as a node. The term, *learn* refers to the task that it needs to perform by considering known, labelled information, Y, and then produce an output for a generalization so that $Y - \hat{Y} \approx 0$ without being specifically programmed. Once a neural network has been trained, it can then be given unlabelled input of the same type as the training data. Generalization from training will be used for projecting one or more answers for the unlabelled information.

With which every incoming connection has a coefficient weight, an activation function which is to convert the input signal as one output for the successive neuron. The weights are the important feature of a neural network as they are the parameters that will get adjusted as learning is being iterated. The sum operation refers to the weighted sum of the inputs that is being calculated for the activation function. The diagram shows two separate processes, first is the summation, after which the total sum is used in the activation function.

Referring to Figure 7, the principal mathematical functions can be explained as a cycle combination. Earlier components for sum and activation are marked as *Operation 1*.

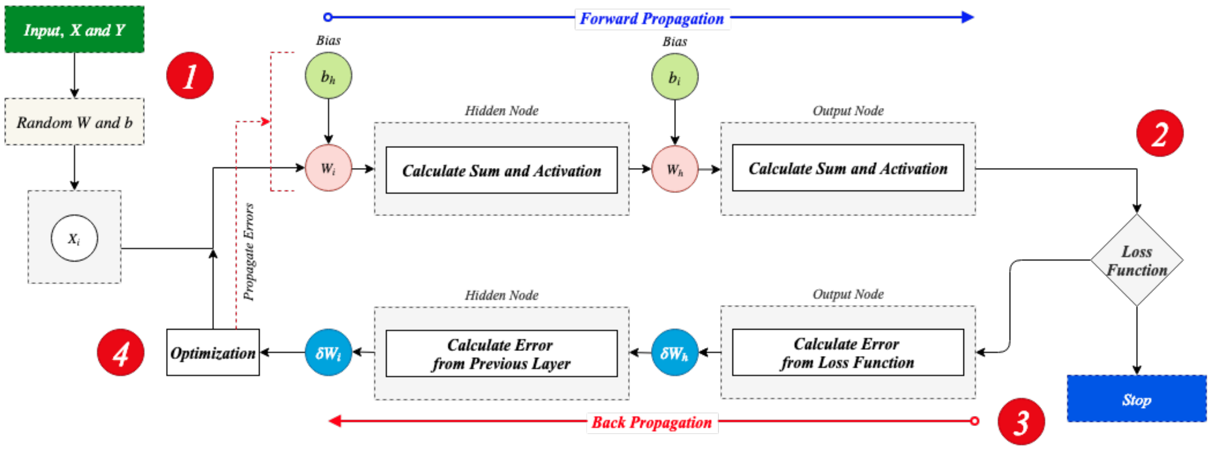


*Figure 7 : Neural network principal mathematical components are numbered as a sequence of operations. **Operation 1** is forward propagation, **Operation 2** is loss function, **Operation 3** is back propagation and **Operation 4** is optimization which includes updating weights and biases.*

By comparing the predicted output against the correct answer in *Operation 2* which is also known as a loss function, corrective measures can be taken by propagating the error backward through *Operation 3*. The process of altering and propagating errors to every connecting weight is then performed in *Operation 4*. Once completed, the cycle will then get repeated. Due to the alteration of connecting weights, the Σ and σ will yield different values, thus affecting the predicted output, $\hat{Y}$for each cycle. The process will continue until the difference between the expected output, Y and predicted output, $\hat{Y}$ is smaller than a certain threshold or until a maximum cycle has reached. Neural network is an adaptive system due to its ability to adjust the connecting weights in order to map between input and output as close as possible [22].

The iterative algorithm works through successive error propagation therefore changing the approximations values for weight coefficients, W. Matrix W can help to predict the class of an example when it is multiplied with the input X. Then add a bias value, b, to the product. Bias value, b, conventionally is a scalar, but for the matrix operation, the scalar is converted to a vector $\vec{b}$ where elements in it is a repeating value of b. The sum of this is then passed through an activation function to produce a value within a certain range which is also referred to as squashing.

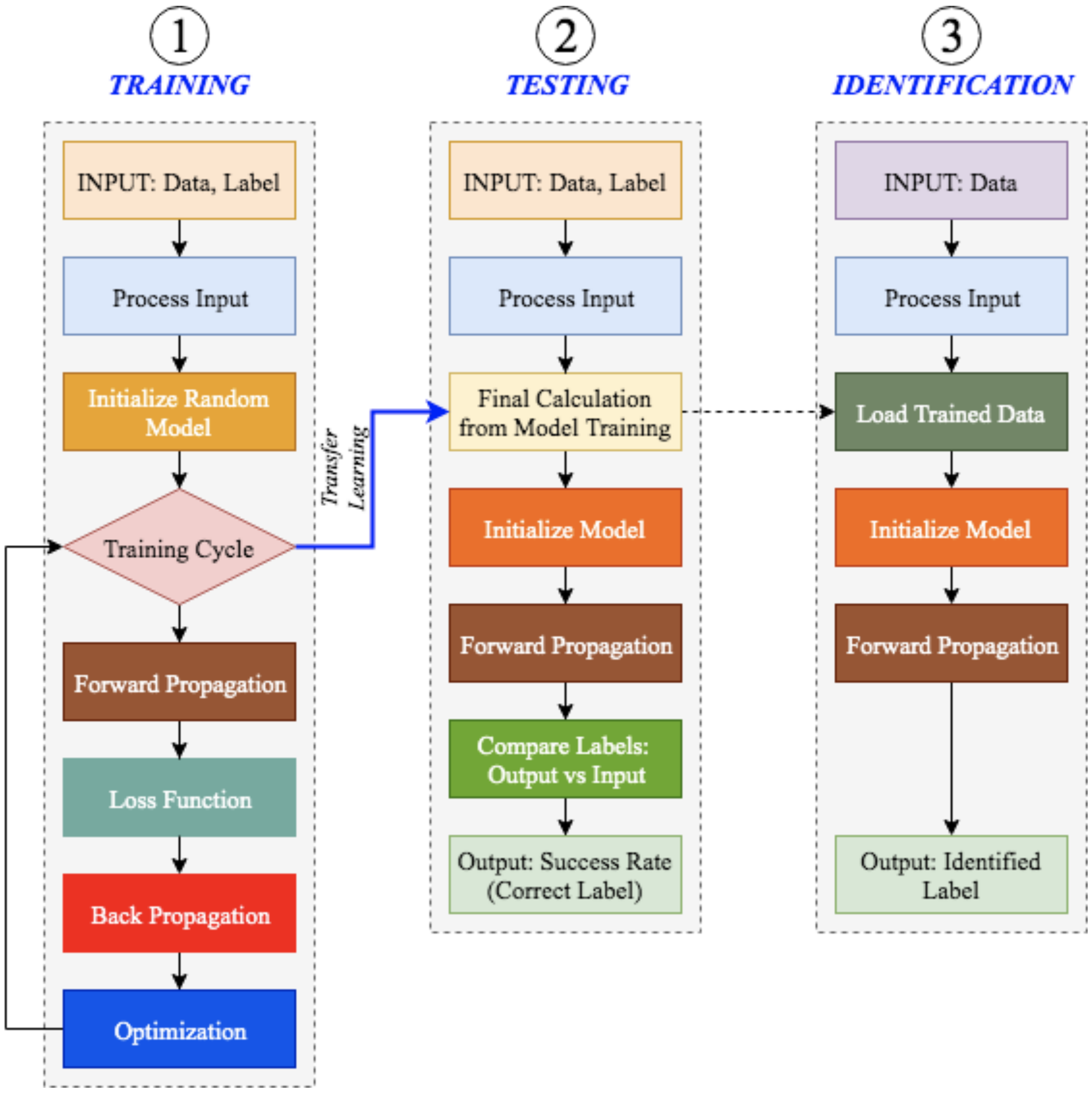


Figure 8 : Three different general processes in neural network but sharing similar predictive model produced from training. The first process is training which has a training loop. The testing process is to validate the result from training. Once it has been validated, then can the model be used for identification.

From Figure 8, the first process is the training which is also the most time consuming and resource intensive and it requires a number of cycles to do forward propagation, error calculation, back propagation and optimization. The time and resources needed will increase based on the total number of the input features and the sample size. The full process does not need to be repeated for verification or identification process. It only needs to be re-run if the architecture of the topology of the neural network has changed, network topology has been altered or dataset has changed considerably. This process is meant to generalize input features and at the end of the process, produce an output in the form of a trained model; which is a collection of adjusted coefficient weights and biases.

The trained model is then used in the second process which is for testing. The purpose of this process is to test the trained model's accuracy rate. If the trained model's accuracy is above a certain threshold it can be used for the final process, identification. The testing process normally takes place after the training process has been completed, therefore the trained model is normally loaded from memory. The purpose for this process is to use the trained model from training process and only using one-time forward propagation to calculate the output. The output given from the model will be compared with known label and the success rate can then be calculated.

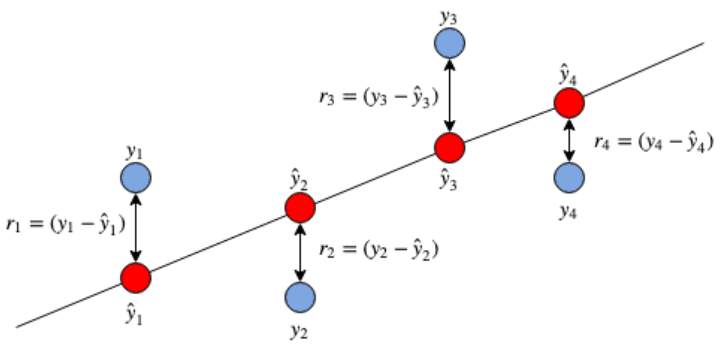
The final process is identification using the saved trained model where the trained model is normally saved into a file. This process will only receive input data and it expects to output a label at the end of the process that can be associated with the input data. Testing and identification are similar in only one part; that is, both are using a one-time forward propagation. The saved trained model can be transferred for use on a different machine or a different programming framework for identification. However, the network architecture and network topology must remain the same.

Also, from Figure 8 above, it can be seen that there is an arrow connecting *Process 1* and *Process 2*. This arrow is referring to *transfer learning* which is a process where a successfully trained model for a particular task on a large scale dataset being repurposed to perform another downstream task without the need to do the training inference all over again [23] [24].

There are different methods to develop a neural network but the most common is *Artificial Neural Network (ANN)* which on its own is very good for clustering, classification, pattern recognition, etc; to add to this, an advancement from ANN is *Convolutional Neural Network (CNN / ConvNet)* which is good for image recognition and computer vision. However, most neural networks will require an additional computational power due to repetitive calculations in order to adjust weight values through forward propagation and back propagation. If the number of neurons is increased or additional layers were added, the computation power and time complexities to compute the calculations is also increased. It is difficult to optimize for both accuracy and complexity simultaneously as neural network calculations are linear and accuracy is based on reducing error through iterations. However, the benefit of neural networks is that the calculated values of optimized weights and biases after a final computation can be saved and later used to identify a newly acquired unlabelled input.

2.2.1 Regression : Linear versus Logistic

Two common algorithms for supervised machine learning are linear regression and logistic regression. Linear regression can be used to produce an continuous *value,* for example time series prediction. On the other hand, logistic regression is for discrete outputs, for example to predict a decision in the form of *yes* or *no* [25] or classification which is non-binary and has more than two decisions. Understanding which algorithm to choose is very important as it shapes how the programming part will be done, for example when building correct label for supervised training. In this project, the supervised learning will be using classification.



y_3
$\hat{y}_4$
$r_3 = (y_3 - \hat{y}_3)$
$r_4 = (y_4 - \hat{y}_4)$
y_1
$\hat{y}_2$
$\hat{y}_3$
y_4
$r_1 = (y_1 - \hat{y}_1)$
$r_2 = (y_2 - \hat{y}_2)$
$\hat{y}_1$
y_2

Figure 9 : Fitting a line through four points and calculating the error between each original value and fitting value, on the regressed line.

The representation of linear regression is an equation that describes linearity to finding the best fitting line between input variables and output variables, $y \approx ax + b + E$ to fit a certain group of data points which can be visualised using a graph with the goal to minimise the vertical distance between all the data points and the regression line which can be seen in the figure above. Regression attempts to predict one dependent variable y and a series of other independent variables x by assuming that the approximation of a linear relationship between x and y can be established when there exists two unknown constants a and b that represent intercept and slope terms in the linear model and E is the error in the approximation [26].

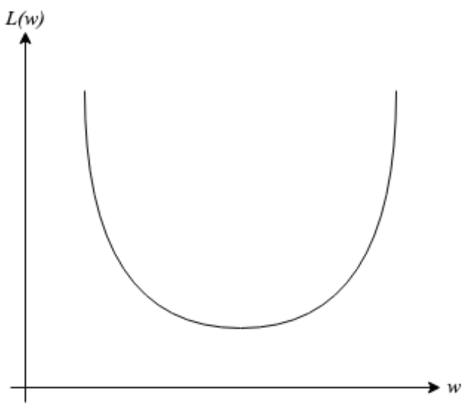


Figure 10 : Non-linear problem fitting.

When data cannot be represented as a linear problem, it cannot be solved using linear regression. Therefore, logistic regression is necessary when solving curves fitting for polynomials or exponentials as in Figure 10. Similar to linear regression, the goal of logistic regression is also to find the values for the coefficients that can weigh each input feature. Prediction for the output is normally transformed using a nonlinear function called the logistic function [27], output variables can be defined as $y \approx a_0 + a_1x_1 + \cdots + a_nx_n$. Logistic regression is superior to linear regression because it can also have output that is linear, therefore it can be applied for both binary and non-binary classification problem.

2.3 COMPONENTS IN NEURAL NETWORKS

A basic neural network can be represented as in Figure 11 below, where the input features are arranged as individual neurons on the left. This is known as the input layer. Each neuron in the input layer is connected to all of the neurons in the hidden layer with each connection having a different coefficient weight. Each layer except the input layer has one bias value. Similarly, each neuron in the hidden layer is connected to all of the neurons in the output layer. The number of neurons in the output layer is determined by the input answers. For a binary answer (*yes* or *no*), one neuron is sufficient. But when the answer is more than one, the neuron should be equal to the answer so that each one can be used to classify each answer.

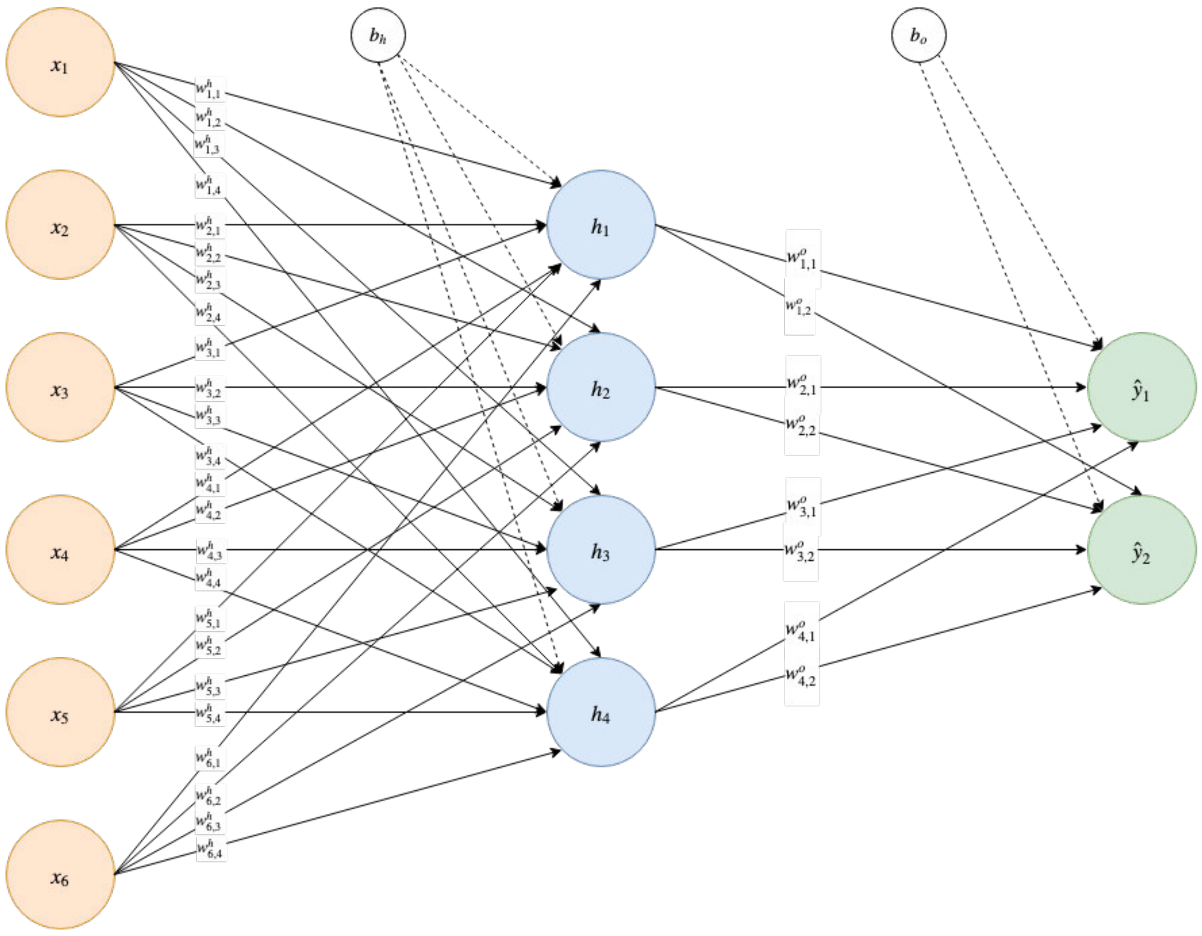


Figure 11 : Fully connected layers in a neural network. On the left is the layer for input, the middle is hidden layer and on the right side is output layer. With exception for input layer, each layer has a set of connecting weights and one bias with exception of input layer.

Information from figure above can be explained further,

1. Input features is the original information to be learnt by the network and must be organized as an independent value,

$$X = \begin{bmatrix} x_1 & x_2 & \cdots & x_6 \end{bmatrix}$$

2. Connecting weights from input layer to hidden layer. Each neuron from input layer is connected to all of the neurons in the hidden layer. Each connection is through multiplication to a certain weight value, designated as w,

$$W^h = \begin{bmatrix} w_{1,1} & \cdots & w_{1,4} \\ \vdots & \ddots & \vdots \\ w_{6,1} & \cdots & w_{6,4} \end{bmatrix}$$

3. Hidden layer is the important part where information is being distilled and generalized being passed to the output layer.

$$H = \begin{bmatrix} h_1 & \cdots & h_4 \end{bmatrix}$$

4. Connecting weights from hidden layer to output layer. Similarly, in number 2 above, each neuron is connected to all neurons in the output layer by a certain weight value.

$$W^o = \begin{bmatrix} w_{1,1} & w_{1,2} \\ \vdots & \vdots \\ w_{4,1} & w_{4,2} \end{bmatrix}$$

5. Output layer is where the prediction about a particular answer is expected.

$$\hat{Y} = \begin{bmatrix} \hat{y}_1 & \hat{y}_2 \end{bmatrix}$$

2.3.1 Input Features, X

Input features should be in the numerical form: an image is normally represented with the colour value between 0 and 255. An input feature of length-n, will have an n number of degrees of freedom for calculations. Too small of n will increase the risk for overfitting and if too large will be take a longer time to converge.

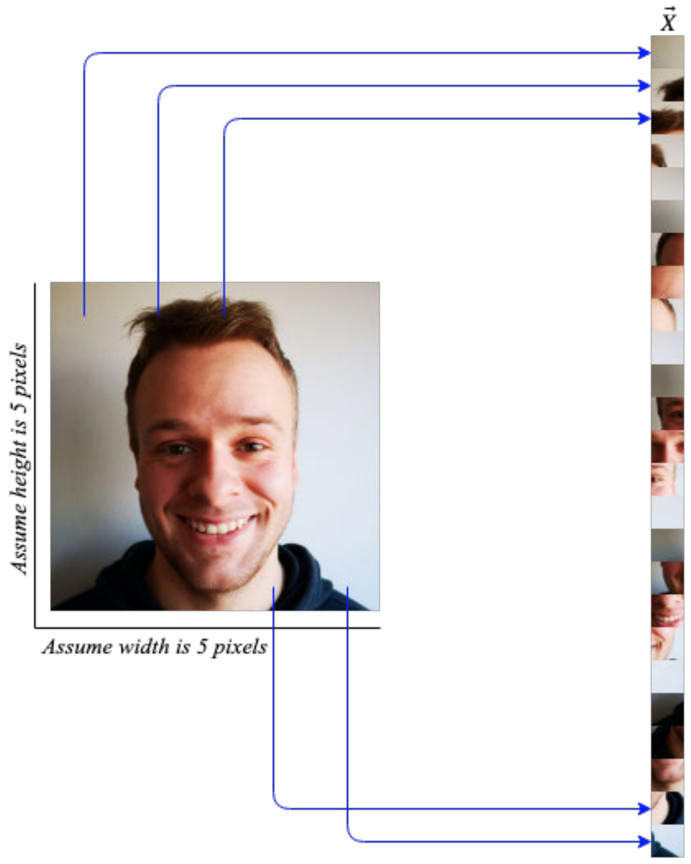


Figure 12 : The process for image flattening can be visualize as above where the data is organized as a row or a column. Assuming that the image is 5×5, then the row (or column) will be 25 in length.

It must also be noted that colour images have more than 1 colour channel and in this research, all of the images in the dataset is using RGB 3-channel colours and therefore when converted to a vector, the way the matrices are converted must also be considered.

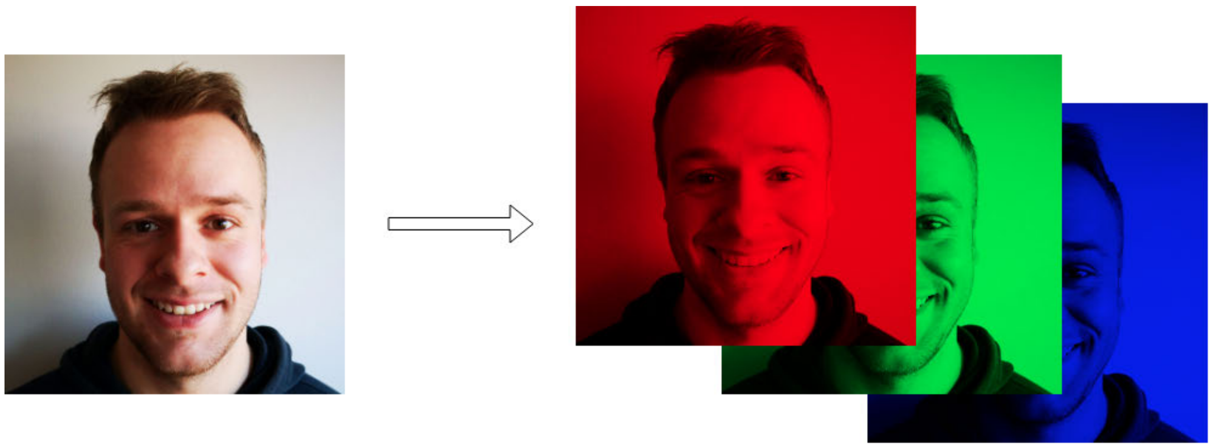


Figure 13 : A colour RGB image has 3 separable colour channels. Each channel provides a set of input length that can be concatenated to form a final input length that will be meaningful for training.

When an RGB image is converted to a vector, each pixel will be flattened into a list of long arrays and then each pixel will be flattened to form the final sequence.

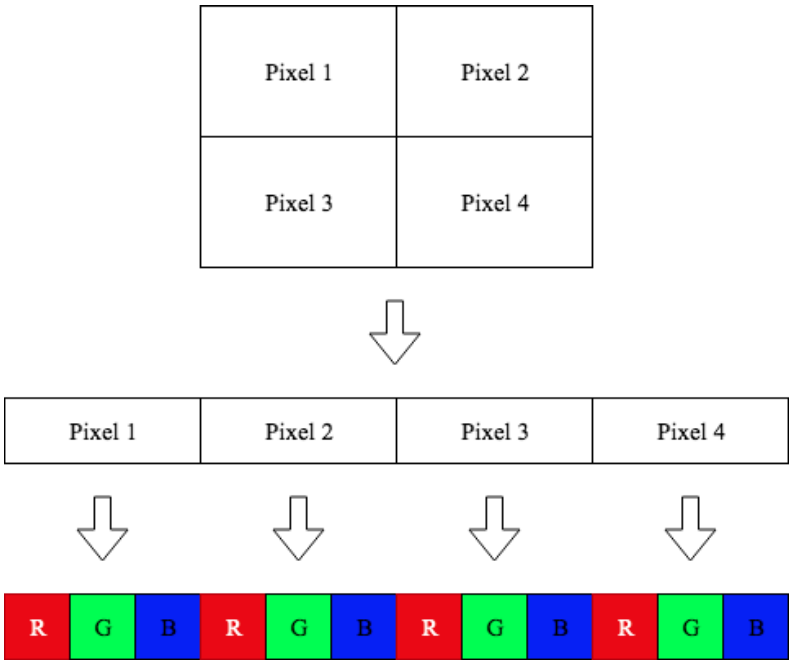


Figure 14 : The transformation of an image to a long vector can be visualized as interweaving colour channels. From n×n×3 matrix, it will then form a new matrix in the form 1×3n^2.

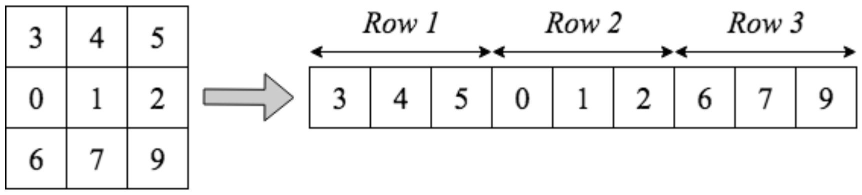


Figure 15 : An example of 3×3 *matrix transformed into a* 1×9 *matrix.*

The information can be represented as a matrix with the dimension of $n \times n$ and then flatten to a matrix with a dimension of $1 \times m$.

$$X = \begin{bmatrix} 3 & 4 & 5 \\ 0 & 1 & 2 \\ 6 & 7 & 8 \end{bmatrix}$$

to,

$$X = \begin{bmatrix} 3 & 4 & 5 & 0 & 1 & 2 & 6 & 7 & 8 \end{bmatrix}$$

Because the input feature is represented as a matrix, multiple inputs can be stacked together. However, each input must have similar dimension of data and must be positioned similarly. For example, when there are 6 more inputs, the vector will be transformed to a matrix and stacked in rows:

$$X = \begin{bmatrix} 3 & 4 & 5 & 0 & 1 & 2 & 6 & 7 & 8 \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} & x_{2,7} & x_{2,8} & x_{2,9} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} & x_{3,7} & x_{3,8} & x_{3,9} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} & x_{4,6} & x_{4,7} & x_{4,8} & x_{4,9} \\ x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} & x_{5,6} & x_{5,7} & x_{5,8} & x_{5,9} \\ x_{6,1} & x_{6,2} & x_{6,3} & x_{6,4} & x_{6,5} & x_{6,6} & x_{6,7} & x_{6,8} & x_{6,9} \\ x_{7,1} & x_{7,2} & x_{7,3} & x_{7,4} & x_{7,5} & x_{7,6} & x_{7,7} & x_{7,8} & x_{7,9} \end{bmatrix}$$

However, to reduce the confusion for matrix calculation as to arrange the input feature similar to Figure 11, the input features matrix can be represented in vertical stacks, transforming the previous X to,

$$X^T = \begin{bmatrix} 3 & x_{2,1} & x_{3,1} & x_{4,1} & x_{5,1} & x_{6,1} & x_{7,1} \\ 4 & \ddots & x_{3,2} & x_{4,2} & x_{5,2} & x_{6,2} & x_{7,2} \\ 5 & \vdots & \ddots & x_{4,3} & x_{5,3} & x_{6,3} & x_{7,3} \\ 0 & \vdots & \vdots & x_{4,4} & x_{5,4} & x_{6,4} & x_{7,4} \\ 1 & \vdots & \vdots & x_{4,5} & \ddots & x_{6,5} & x_{7,5} \\ 2 & \vdots & \vdots & x_{4,6} & \ddots & \ddots & x_{7,6} \\ 6 & \vdots & \vdots & x_{4,7} & \ddots & \ddots & x_{7,7} \\ 7 & \vdots & \vdots & x_{4,8} & \ddots & \ddots & x_{7,8} \\ 8 & x_{2,9} & x_{3,9} & x_{4,9} & x_{5,9} & x_{6,9} & x_{7,9} \end{bmatrix}$$

A feature matrix which contains an m-samples with an n-features can be expressed as $X \in \Re^{m \times n}$.

2.3.2 Layers

A basic topology of a neural network consists of mandatory one input layer, a minimum of one hidden layer and one output layer. Successive layers are similarly fully connected by a set of weights.

Neurons in the input layer are kept constant throughout iteration, completely and uniquely determined based on the training data features. It must be noted that input features have no predecessor neurons, therefore it has no incoming weights which means that back propagation stops when it reaches the input layer. Certain neural networks use a different configuration by adding one additional node for a bias term. If data feature X is in a vertical shape, the bias term can be added as the first row for all data features.

The purpose of the hidden layer is to transform the inputs into something that the output layer can use, by transforming data from the previous layer into a *"higher-level"* representation of that data. *"Higher-level"* means that current layer contains a compact and more salient representation of the original data [28]. There is no fixed method for picking the number of hidden layers which depends on the type of activation functions, type of problem linearity and the number of output [29]. According to Jeff Heaton [30], picking the hidden layer density can be illustrated as in Table 2 below. In theory, there is no limit to the number of hidden layers but one literature [31] indicates that a maximum of five layers (one input layer, three hidden layers and an output layer) can be used to solve problems of any complexity; however, most technical projects reported success when using only three layers.

Density	**Result**
0	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

Table 2 : Hidden layer density and what the expected result be.

The problem with a hidden layer is not only the density but also the number of neurons in each hidden layer. Proper size must be carefully considered as it has a tremendous influence on the final output. Too small will result into underfitting, a problem where identification is bad, which was due to the size of neurons in the hidden layers are inadequate to detect important signals in a complicated dataset. On the opposite spectrum, too many will result in overfitting, a problem where identification is good for training but poor for new data; when the neural network has too much information for processing but the input features has a limited amount of information that is not enough to train all of the neurons in the hidden layer. It leads the network to be a memory bank that can recall the training set to perfection, but does not perform well on samples that was not part of the training set [32]. Even when the input features and dataset are big enough to avoid underfitting and overfitting, respectively, but due to the large number of neurons, the calculation will increase time complexities and it would be impossible to train a neural network properly [30]. A suggestion to fix the number of hidden layer based on the known size of input and output layer, to avoid overwhelming a neural network with too much information, Geometric Pyramid Rule was introduced in 1993 [33]. It suggests that the size of a hidden layer should be about the value of square roots of a product of multiplication of the size of the input layer and size of the output layer. The Geometric Pyramid Rule seems to be appropriate in the context of this project. Its mathematical formula goes as follows:

$$L_h = \sqrt{L_i \times L_o} \qquad \textit{Equation 1}$$

The purpose of output layer is to transform the hidden layer activations into a scale within an expected answer. Similar to the input layer, every neural network has exactly one output layer. Determining a proper size for this layer is fairly simple because it is completely determined by the chosen learning algorithms model. Linear regression returns only a single value; therefore, the output layer can be set to have 1 neuron. However, for classification problem, it can either have 1 neuron if the answer is binary but if it is classification, then the number of neurons equals to the number of classification label.

There are two mathematical operations involved in each of these layers. First calculation is the product of input features and weights which is to compute a linear transformation from the previous layer. After that, the sum of the product will be passed to a function to produce a non-linear value. The non-linear value then will be the input for subsequent layer's and the same product and transforming to non-linear value process starts all over again. This concept of transforming product value into a new meaningful set of value is the heart of machine learning, and the primary capability of neural networks [34].

Forward propagation is a *score function* that maps features to classification scores. It maps the pixel values of an image, D, to confidence scores for each classification, K. Function dimensionality reduction is within real values, $f : \Re^D \mapsto \Re^K$ for its sum function:

$$f(x_i, W, b) = W^T \cdot x_i + b$$ *Equation 2*

Where i lies within the space for the total sample of m and each sample i in m has an equal dimension $D = (I_W, I_H, I_C)$ that represents an image's width, height and colour channels respectively which must be transformed into a feature vector N with a dimension formed from D in the terms $I_W \times I_H \times I_C$; the length for sample i is $i = 1, \cdots, m$. For K, it is the subspace for total number of distinct classification scores which exists in $y_i \in \{1, \cdots, K\}$ for each sample i [35].

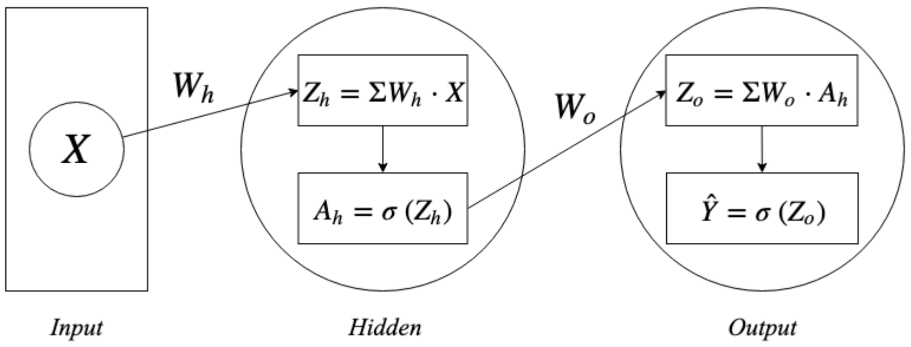


Figure 16 : Considering a topology of 3-layer neural network; the mathematical concept for forward propagation can be explained as a linear flow, where the sum for a particular layer (either hidden or output), is the product from previous layer's value and its corresponding weight. This sum value is then passed to an activation function to produce a new value either for the layer's sum calculation or to produce the final output prediction.

In Figure 11 above, the function Z_h maps the input layer to the hidden layer. The subsequent Z_o maps the hidden layer to the output layer. Therefore, $\hat{Y}$ can be calculated by forming these two functions $\hat{Y} = Z_o(Z_h(X)))$ If the level of complexity per function is limited, then $Z_o(Z_h(X))$ can compute things that Z_o and Z_h can't do individually. In terms of formal calculation,

$$\hat{Y} = \sigma(\sum(W_o \cdot \sigma(\sum W_h \cdot X))) \qquad \textit{Equation 3}$$

2.3.3 Weights

Weights coefficient matrix are initialized randomly with different and non-zero values to avoid linear calculation and also to avoid having too small or too large values that affect signal strengths. Individual weights represent the strength of connections between neurons in two separate layers and these values influence the outcome in the subsequent layer. One method for initialization is to use *Xavier Initialization*, proposed by Xavier Glorot and Yoshua Bengio, based on Gaussian distribution [36] and is defined as:

$$W \sim \left[-\frac{\sqrt{6}}{\sqrt{n_{layer} + n_{layer+1}}}, \frac{\sqrt{6}}{\sqrt{n_{layer} + n_{layer+1}}}\right]$$

The weights can be arranged in a matrix where the number of rows represents the number of neurons in a previous layer, and the number of columns represents the number of neurons in a current layer. Using the previous example, let's say the current layer has n neurons:

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & w_{2,3} & \iddots & w_{2,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ w_{9,1} & w_{9,2} & w_{9,3} & \cdots & w_{9,n} \end{bmatrix}$$

The intuition is that the greater the number of weights, the more different mapping functions a network can fit, and therefore it will also increase the degrees of freedom and effective time complexity [37], however the size of weights are determined by the size of each layer and layer densities.

2.3.4 Bias

The main function of a bias is to provide every node with a trainable constant value but without interacting with the actual feature $X = \{x_1, x_2, \cdots, x_n\}$. It allows a linear regressor, $y = ax_i + b$ to move the line up and down to fit the prediction with the data points better. Without b the line always goes through the origin $(0, 0)$ and the result may be of a poorer fit. The value for b is normally within $[-1, 1]$ except for 0.

For the sake of computation, bias b which is normally represented as a scalar can be represented as a vector with the same size as the current layer, with the same value b in each row.

$$\vec{B} = \begin{bmatrix} b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ \vdots \\ \vdots \\ b_{n-1,1} \\ b_{n,1} \end{bmatrix}$$

2.3.5 Target Output for Classification

For a supervised training, the goal is to discover a certain rule to recreate the output by mapping the input features to the target output. Therefore, a neural network must be provided a set of ground truth labels, a technical term for *correct answers*, in order for it to perform classification successfully, similar types of labels are grouped in classes [38]. A target output for each input is provided so that predicted output can be compared against a correct answer after each iteration. This comparison is done by means of a loss function. Each class prolongs a correct label which can be represented as a vector. Class label can be in the form of repeated binary numbers $\{0, 1\}$ for binary classification, but multiclass classification instead requires a range of positive integers $\{0, 1, 2, 3, \ldots, n\}$ or any other real numbers, $\{-\frac{n}{i}, \ldots, \frac{j}{k}\}$ where $i, k \neq 0$ and $n, i, j, k \in \mathbb{R}$

Extending from the example that has 7 classes, the correct label must also have a length of 7 elements.

$$Y = \begin{bmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} & y_{1,5} & y_{1,6} & y_{1,7} \end{bmatrix}$$

Multi-classification must transform the expected output to a binary form using one-hot vector encoding, which is $1 - of - K$ encoding for K number of classification, where the target is 1 at the index of the class for the correct classification, and 0 everywhere else. An example with 10 outputs, the target vector would be $(0, 0, 1, 0, 0, 0, 0, 0, 0, 0)$ if the target is the third in the classification.

2.4 MATHEMATICAL COMPONENTS

For the purpose of this research, the mathematical components will only focus on the important functions that will make a basic neural network able to perform.

2.4.1 Forward Propagation

Forward propagation as well as backpropagation was developed in the 1970s and the process was written by Paul Werbos in his PhD thesis in 1974 [39]. Feedforward neural networks are also known as *Multi-layered Network of Neurons (MLN)*. These networks of models are called feedforward because the information in the neural network only travels forward; from input neurons through hidden layers (a single one, or many layers) to output neurons.

Input features ***x*** are represented by a vector ***X***,

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \end{bmatrix}$$

Weights ***w*** are represented by a matrix ***W***,

$$W = \begin{bmatrix} w_{1,1} & \cdots & w_{1,j} \\ \vdots & \ddots & \vdots \\ w_{i,1} & \cdots & w_{i,j} \end{bmatrix}$$

The sum for one layer, Z_i, is calculated using output of the previous layer, $X = A_{i-1}$, and the connecting weights, W_i. The formula for sum function is as follows,

$$Z_i = W_i^T \cdot X_{i-1} + b$$

A schematic diagram in Figure 17, locates mathematical functions necessary for forward propagation, in places in the neural network where they belong.

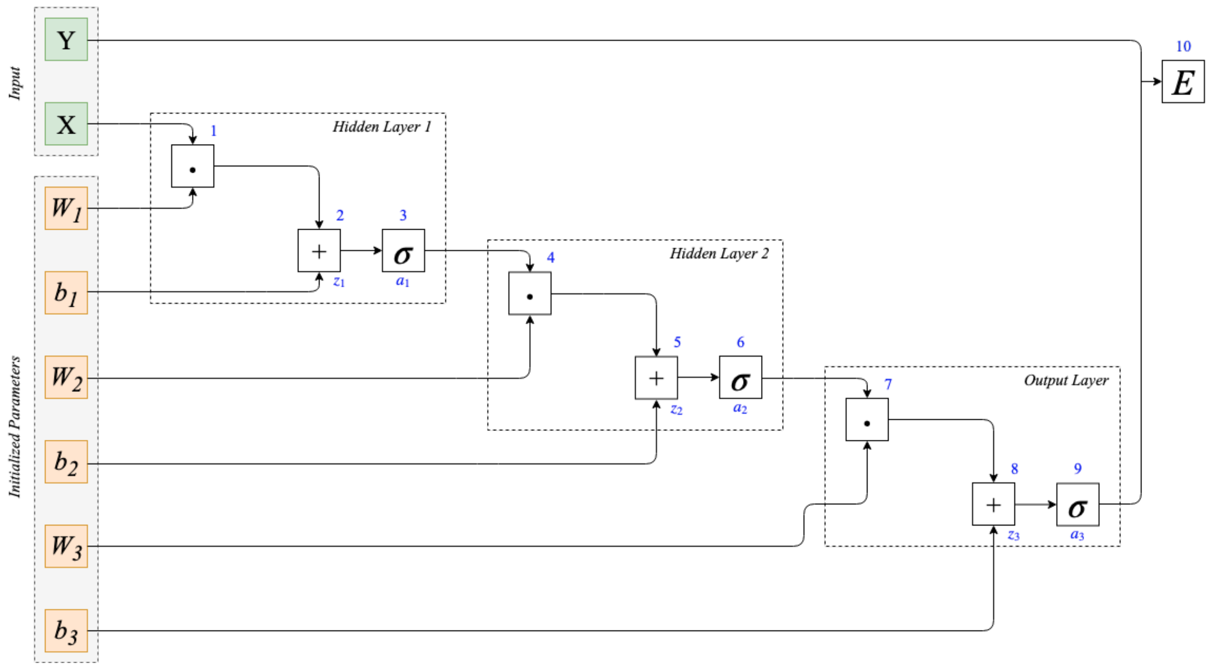


Figure 17 : Computation graph of mathematical operations for forward propagation. In this diagram, addition, squashing and dot product of matrices are represented. These operations are repeated in each layer.

2.4.2 Activation Function

Activation function reduces the input into a range of permissible output, for example a range within a certain upper and lower limit. The output works as an input for the next layer, or in case it is the output layer, it produces values within the target output [40].

i. Sigmoid

This research is interested in sigmoid and its extension, Softmax, as it can produce better output for classification considering that the expected outputs are non-binaries. Sigmoid is a two-class logistic regression and due to its denominator, the expected output $y_i = \sigma(z)$ is limited to $y_i \in \{0, 1\}$.

$$\sigma(z) = \frac{1}{1+e^{-z}}$$ *Equation 4*

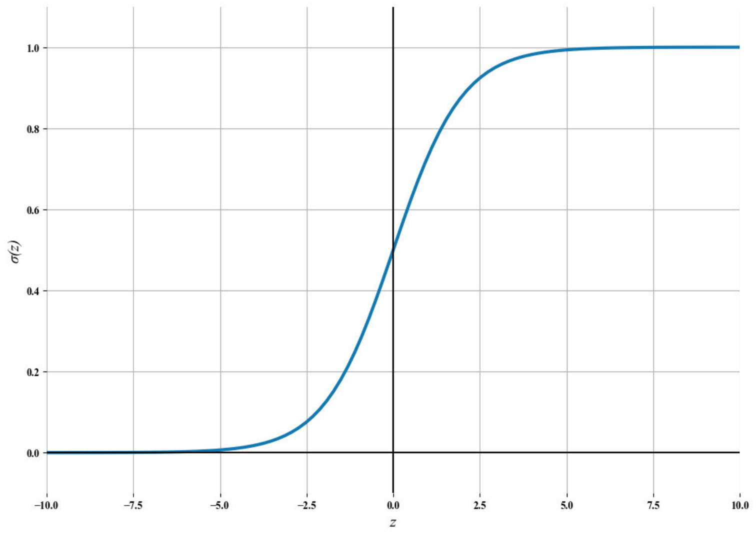


Figure 18 : Sigmoid function when plotted on a graph. Gradient can be seen to vanish when $x < -5$ and $x > 5$.

Sigmoid function has its own problem, as when the sum is too big or too small, it then still produces value which is close to 0 or 1 and this is called vanishing gradient problem which would lead to problematic gradient calculations as sigmoid output saturates [40].

ii. Softmax

Softmax is a generalisation extension of the sigmoid logistic regression and known as multinomial logistic regression, which is used when there are more than two, $k > 2$ classification to predict. Each element, y_i in output $\hat{Y} = y_1, \cdots, y_i, \cdots, y_k$ will be between 0 and 1. The denominator in *Equation 5* ensures that the k outputs will sum to 1. This function minimizes the cross-entropy between the estimated classification probabilities - in the *"true"* distribution, all probability mass is in the correct class, i.e. $p = [0, \cdots, 1, \cdots, 0]$ contains a single 1 at the y_i-th position.

$$\sigma(z) = \frac{e^z}{\sum e^z}$$ *Equation 5*

iii. Step Function

A simple activation function called *a step function* has a certain threshold, and if the input value is lower than the threshold, the output is 0, otherwise the output is 1. Such a function changes the result of the sum function into a non-linear output. But the step function is not differentiable at zero, therefore it can’t be used for backpropagation [41].

$$Step(z) = \begin{cases} 0, z < 0 \\ 1, z \geq 0 \end{cases}$$ *Equation 6*

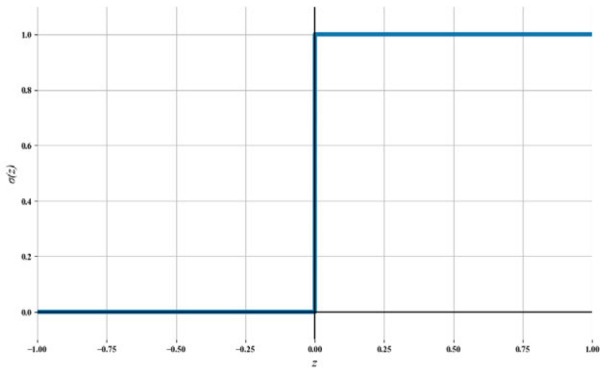


Figure 19 : Plotted graph of a step function.

iv. Hyperbolic Tangent

Hyperbolic tangent (tanh) is a zero centred function with an output in the range of $y_i \in \{-1, 1\}$. It is more practical than sigmoid but still it suffers from the vanishing gradient problem.

$$tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$ *Equation 7*

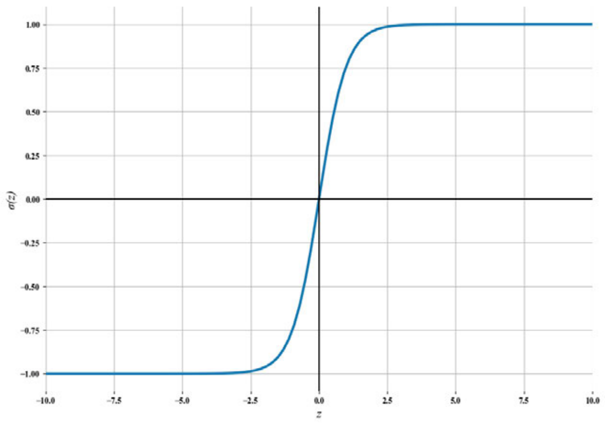


Figure 20 : Plotted graph of hyperbolic tangent.

v. ReLU

Rectified linear units (ReLU) was proved to be 6 times better at converging compared when using hyperbolic tangent. It is so because ReLU does not have an upper bound and thus it solves the vanishing gradient problem (i.e. when a function is not sensitive enough to values close to the limit of an interval) [42]. However, it has its own limitations as it cannot be used for an output layer, and if it generates the value of 0, it will result in a dead neuron which cannot be activated again in the next cycle of calculations [40].

$$ReLU(z) = max(0, z) = \begin{cases} 0, z < 0 \\ z, z \geq 0 \end{cases}$$ *Equation 8*

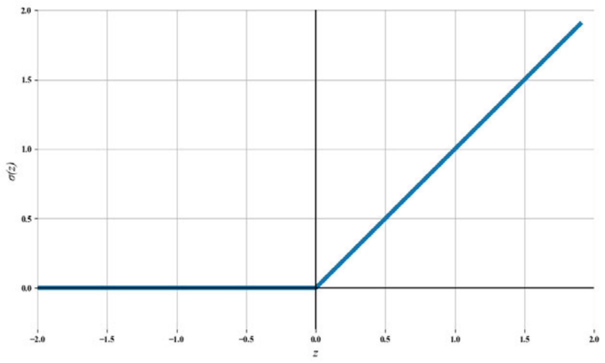


Figure 21 : Plotted graph of ReLU.

vi. Asymptote Problem with Sigmoid and Softmax

Boundaries can be set before certain values can be used by either Sigmoid or Softmax or their respective derivations. This is done by choosing a particular, very small value for a variable *epsilon* for example 10^{-8}. Then, any value smaller than *epsilon* is set to *epsilon*, and any value bigger than (1 – *epsilon*) is set to (1 – *epsilon*). This prevents the activation during forward propagation or derivation during backpropagation from dealing with asymptotic values or other issues. That could otherwise happen if numbers the calculations are started with, are very close or equal to limit values of the interval $[0, 1]$ [43].

2.4.3 Loss Function

This purpose of loss function is to measure how close the output from a network is from the target output which is termed as *network error* which can provide a salient indicator of the output performance for calculation in the current stage [44]. Loss function is expressed between a *true* distribution of target output, Y and an estimated distribution of predicted output, $\hat{Y}$ is expressed as $E = L(Y, \hat{Y})$ to quantify the agreement between the predicted outputs and the expected outputs with the purpose to define the details and measure how far the output is from the expected outcomes and the expected direction when the output $\hat{Y}$ is close to Y should come close to 0. Accumulated loss value for every calculation can be plotted onto a graph to study how the network is performing as the main goal is to reduce the error and a graph can quickly tell whether the errors are converging, diverging or unstable.

Cross-entropy function is used for multi-classification. For a two class problem (C_1 (target $y = 1$) and C_2 (target $y = 0$)), then the output can be interpreted as [45]:

$$\hat{y} \sim P(C_1|x) = P(y = 1|x)$$ *Equation 9*

(read: $\hat{y}$ is approximately equal to the probability of C_1 given x, which is equal to probability of $y = 1$ given x)

$$(1 - \hat{y}) \sim P(C_2|x) = P(y = 0|x)$$ *Equation 10*

(read: $(1 - \hat{y})$ is approximately equal to the probability of C_2 given x, which is equal to probability of $y = 0$ given x)

General equation for cross-entropy can be written as [46],

$$L(Y, \hat{Y}) = CE = -\frac{1}{m} \sum_{i=1}^{m} Y_i \, log(\hat{Y}_i) + (1 - Y_i) \, log(1 - \hat{Y}_i)$$ *Equation 11*

For multi classification which has more than two outcomes, the cross entropy of probability distribution can be simplified as follows,

$$L(Y,\hat{Y}) = CE = -\sum Y \cdot log(\hat{Y}) \qquad \textit{Equation 12}$$

Loss function can help to evaluate the performance of a neural network visually, as can be seen in the following Figure 22, 23, and 24.

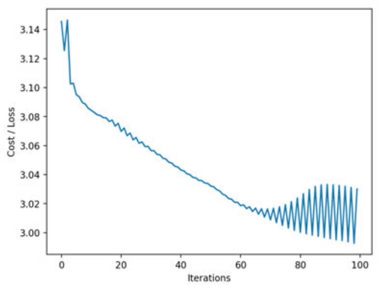


Figure 22 : Oscillating

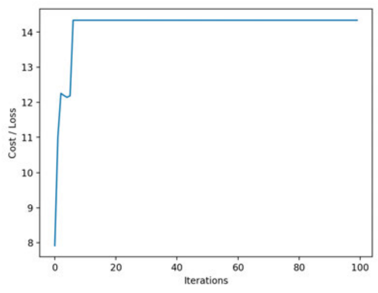


Figure 23 : Diverging

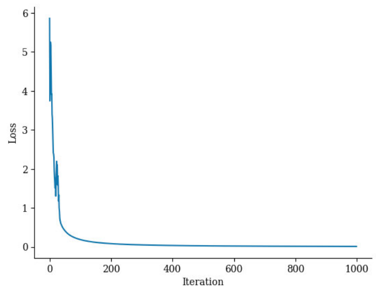


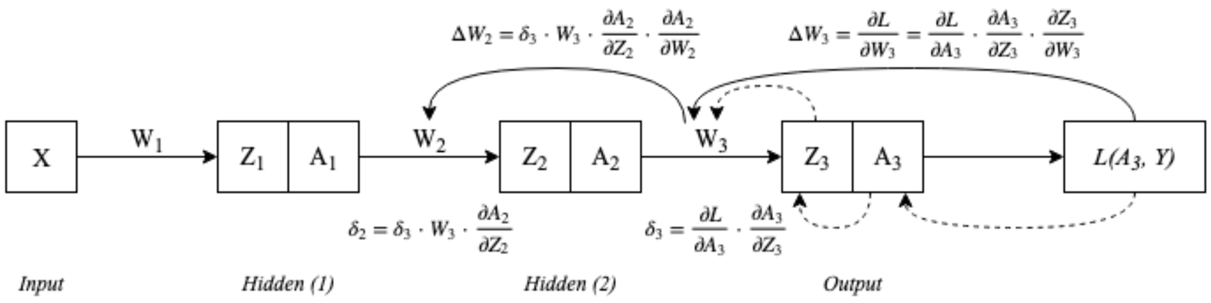
Figure 24 : Fitting and converging

2.4.4 Backpropagation

The goal of backpropagation is to compute the partial derivatives $\frac{\partial L}{\partial W}$ and $\frac{\partial L}{\partial b}$ of the loss function $L(Y,\hat{Y})$ with respect to any weight W or bias b in the network [47]. One cycle through the entire training dataset is called a training epoch. Backpropagation should stop after a fixed number of epochs, or when the error is smaller than a given threshold.

The possibility to do backpropagation lies on the chosen activation function, if it is a continuous function then it is differentiable. If it is differentiable, then it will give out a certain value at a given point that can be used to minimize coefficient weights [48].

The backpropagating process can deploy a variant of the *Delta Rule*, which starts by calculating the error residuals, L, which is the difference between the actual outputs, $\hat{Y}$, and the target outputs, Y. Using this error value, connecting weights are then increased or decreased in proportion to the error multiplied by a scaling factor, α. The complex part of this learning mechanism is for the network to determine which input contributed the most to an incorrect output and how does that element get changed to reduce the error. The process is backward process which started from the output layer and then move towards hidden layers before reaching the input layer in a linear sequence.



$\Delta W_2 = \delta_3 \cdot W_3 \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial A_2}{\partial W_2}$
$\Delta W_3 = \frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial W_3}$
X
W_1
Z_1
A_1
W_2
Z_2
A_2
W_3
Z_3
A_3
$L(A_3, Y)$
$\delta_2 = \delta_3 \cdot W_3 \cdot \frac{\partial A_2}{\partial Z_2}$
$\delta_3 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3}$
Input
Hidden (1)
Hidden (2)
Output

Figure 25 : A neural network with 4 layers, which has an input X, with each layer has connecting weights, W_n; sum of a particular node, Z_n; activation of a particular node, A_n; and loss function, L. The backpropagation started from the last process and move backward as shown by the arrows. The residue loss from function L will affect the value for Δ and δ as the process traverse backward.

Procedure for the calculations to apply a delta rule process can be seen in the Figure 25 over how to calculate proportional error and turn it into the terms of the delta rule in order to adjust connecting weights [49], [50] and the figure above is visualizing a neural network with 4 layers. Backward calculations are marked with Δ and δ.

After the value for $L(Y, \hat{Y})$ has been obtained, then the Δ for the fourth layer, which is also the Output Layer,

$$\Delta W_3 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial W_3}$$ *Equation 13*

The Δ for Hidden (2) Layer is by calculating from the Output Layer

$$\Delta W_2 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2}$$ *Equation 14*

The Δ for Hidden (1) Layer, it is by calculating from the Output and Hidden (2) Layers,

$$\Delta W_1 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A_1} \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1}$$ *Equation 15*

It can be seen that some variables are repeating and therefore can be factored to produce the error δ. From the equation above, the partial difference for Loss Function and Activation Function are taken out produce the error δ below,

$$\delta_3 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3}$$ *Equation 16*

Observing *Equation 14, 15* and *16*, the error, δ for layer j can be generalized into,

$$\delta_j = \frac{\partial L}{\partial A_j} \cdot \frac{\partial A_j}{\partial Z_j} = \frac{\partial L}{\partial A_j}\sigma'(Z_j)$$ *Equation 17*

and when calculated as a matrix, it must use Hadamard product which is an element-wise product of two vectors,

$$\delta_j = \frac{\partial L}{\partial A_j} \odot \sigma'(Z_j)$$ *Equation 18*

and the δ term can generalized into *Equation 17* when it is being implemented as a loop function,

$$\delta_j = ((W_{j+1})^T \delta_{j+1}) \odot \sigma'(Z_j) \qquad \textit{Equation 19}$$

2.4.5 Optimization

The goal of optimization is to alter weights coefficient and biases parameters in order to minimize the loss function of the model on the training dataset with respect to the parameters of the accuracy function. It works by calculating the errors between the prediction and expected answer and applying these errors by updating the parameters on each iteration so that the predicted output is as close as possible to the correct answers. A few known optimization algorithms are Adam Optimizer, Stochastic Gradient Descent (SGD), Momentum Gradient Descent, Mini-Batch Gradient Descent and the most well-known, Full-Batch Gradient Descent. An illustration of how gradient descent works is presented in Figure 26.

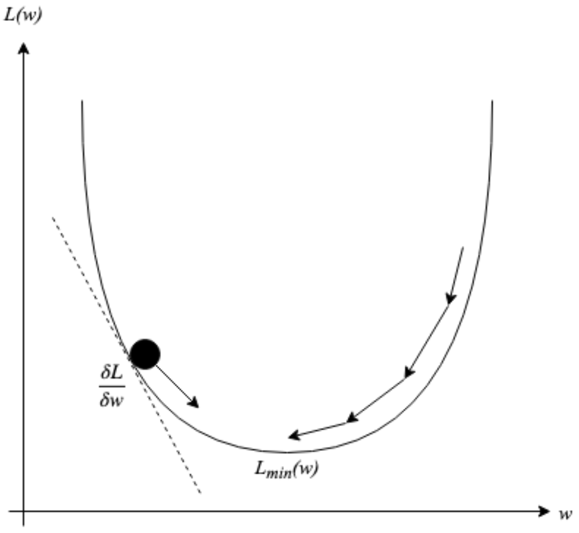


Figure 26 : Schematic of gradient descent

Stochastic gradient descent modifies the weights after every training of every sample. It means that forward and back propagations take place for every sample, hence these processes increased the training time. The process can produce a quick result during the early process but then will start to be noisier as training progresses. The process updates individual weight,

$$W \leftarrow W - \alpha \cdot \frac{\partial E}{\partial W}$$ *Equation 20*

i. Momentum Gradient Descent

Momentum gradient descent is built upon stochastic gradient descent which includes one more useful feature called *momentum*. Momentum in a way, works like a memory, it accumulates all past directions of the gradient descent. This way, momentum gradient descent is less prone to get stuck in local minima of a function because the descent would not stop right when the minimum is reached. The momentum can be imagined as a heavy ball that keeps rolling even if it encounters a bump on the road. Weights are always updated by a new value of momentum instead of a value that comes directly from gradient descent.

ii. Learning Rate, α

Learning rate, α is a coefficient value to control the speed of changing rate during the learning process by affecting the gradients. If the learning rate is too large, gradient descent will overshoot the minima and diverge as the speed is too fast. If the learning rate is too small, the algorithm will require too many epochs to converge because the speed is too slow and can become trapped in local minima more easily. The chosen value for learning rate α is in theory between $[0, 1]$ but in reality is almost always very close to zero.

iii. Regularization Constant

Regularization constant is part of the backpropagation mechanism that is there to reduce issues like overfitting. By preventing that, the neural network suffers less from a generalization error which arises when the neural network is faced with a part of the dataset that was not used for its training. If some kind of regularization is included, the neural network should exhibit smaller discrepancies between training accuracy and testing accuracy [51].

2.5 ARTIFICIAL NEURAL NETWORK

Artificial Neural Network (ANN) is a computational nonlinear model and consists of multiple layers and often referred to as Multi-Layer Perceptron (MLP). ANN is good for adaptive learning because it has the ability to complete a new task based on data given during training and for problem that cannot be solved by separating classes with a straight line. The architecture consists of a group of neurons that form a layer for input, a hidden layer and an output layer.

The input layer is where the input is projected into a network and each neuron carries different values and weights. The hidden layer is also known as *Distillation Layer* as this is where information is being extracted as the name suggests, being distilled, for important features from the input layer by removing any redundant information. Information from the hidden layer will then passed to the output layer, where there can be more than one output. Every neuron in input layer is connected to every neuron in hidden layer. If there exists n-number of neurons in hidden layer, there will be n-weight of connections for each neuron in the input layer. Similarly, for neurons from hidden layer to output layer.

While there can only be one input and output layer, there can be more than one hidden layer. The purpose of hidden layers is to accept pre-processed values from previous layer and refine the information with specific logic, which means more information is being distilled. By adding more hidden layers, a network can be made more flexible and works better to model a complex nonlinear relationship. A network with more than one hidden layer is called deep neural network. Neurons computation in ANN can be computed using parallel computation paradigm, therefore if a network is large, at a particular stage in the network can be split up and distributed for parallel computation. Once a neural network has been trained, the final fitted weights in the hyperparameter can be interpolated to continuously recognise unknown input patterns.

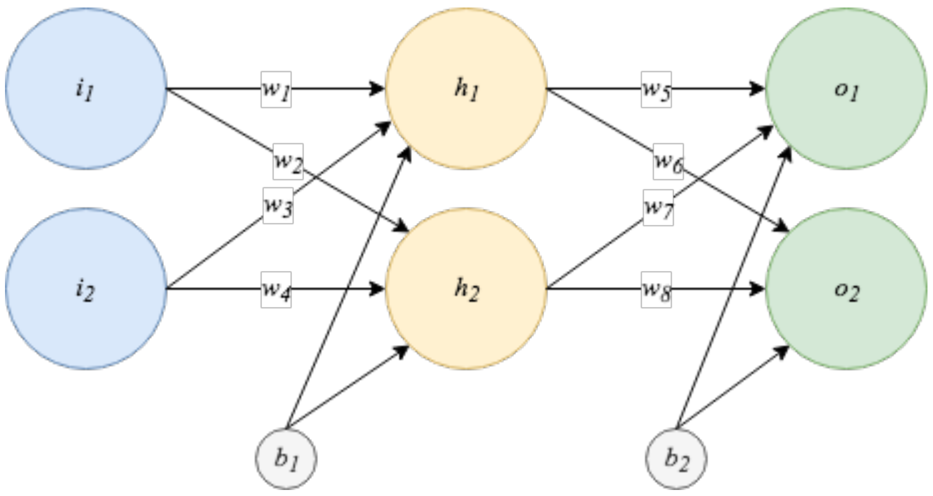


Figure 27 : Output can be more than one neuron. Neurons in hidden layer accept a summation values from each node in input layer. Then the value will be processed with a certain specified logic before being passed to all of the neurons in the successive layer.

The process for ANN can be explained as,

1. Initialize random weights and bias for a neural network topology.

2. Feed forward by passing the input through the network layers and calculate the output for each layer $l = 2, 3, \cdots, L$ compute $Z_l = W_l^T A_{l-1} + b_l$ and $A_l = \sigma(Z_l)$

3. Calculate loss function which is the absolute error on the performance difference between the desired output and the actual output for current model, $L(Y, \hat{Y})$.

4. The objective is to minimize the loss function by optimizing the weights and to make decisions on weight-update. In mathematics, the derivative of a function at a certain point, gives the rate or the speed of which this function is changing its values at this point. By checking the derivative value; if it is positive, the error increases if when weights are increased, then the decision would be to decrease the weight. If it is negative, the error decreases if when weights are increased, then the decision would be to increase the weights. If it is 0, then it has reached its stable point.

5. Calculate ΔL which is the rate of change of loss with respect to the output from the activation function.

6. Back propagation: The error contribution of each neuron is calculated starting from the gradient of the loss function, and the same is back-propagated to the previous layer.

7. Update and optimize the weights based on values calculated using the gradient calculations.

8. Iterate the same steps from 1–7 till the model converges by checking the minimized loss function is smaller than a certain threshold or until the maximum number of iterations.

2.6 TRAINING

In a training phase, neural networks scrutinize a set of images with known label, the learning process is an iteration of a process to extract important information so that the prediction will come as close as possible to the answer label. After the training process is complete, the *trained model* can be saved into an external file. This model is then provided for the face recognition program to work on. A training problem can be explained in terms of error between output from the network against its corresponding target with the aim to minimize the error function, $E = L(\hat{Y}, Y)$ [52]. A training model *learning* process can be visualized by plotting accumulated loss value from each epoch and the graph should decrease as epoch grows.

2.7 TESTING

The purpose of testing is to gauge the trained model accuracy whether training is optimized, underfitting or overfitting by using dataset when has not been used for training to see how the trained model behaves. Overfitting occurs when a model performs well within a specific training set and minimizes the error but when presented with data which belongs to any classification in the previous training, the error diverges. It often occurs because a network appears to have successfully learned but in reality, only *memorized* the initial training set. From a technical perspective, it occurs when a network took too much attention to unnecessary details (which is also referred to as *noise*) instead of focussing on the important signal and can be analysed by comparing the accuracy values from training and testing are diverging. Some solutions to overfitting are by using regularization, data augmentation or increasing the training data. Underfitting is the opposite situation from overfitting which is caused by too few features or samples during the training [53].

2.8 EVALUATION

Overfitting occurs when training went *too well* and fit too closely to the used training dataset. This could be due to the higher complexity of features of the dataset but from a small sample where it leads to high accuracy during training but gives out smaller accuracy when using new unseen data [54]. On the opposite spectrum is underfitting where the model does not fit the given training data therefore it yields poor accuracy during training and also when predicting unseen data. It probably is due to low number of features and also due to smaller number of samples [54].

False-positives is when a neural network gives out a predicted output within a known label when it should have given a feedback that no match exists. A false-negative occurs when the neural network misses a match it should have made.

2.9 SUB-CONCLUSION

The fundamental mathematical concepts for neural networks have been explained in depth to cover forward propagation, loss function, back propagation and optimization in order for the most basic Artificial Neural Network (ANN) to be implemented to solve logistic regression problem. Hence, a number of possible case studies for this chapter are:

1. Keeping the topology of ANN for only 3 layers and also using Softmax activation for output layer but alternating activation function for hidden layer to employ ReLU, tanh and sigmoid; what can be observed in terms of time and accuracy?

2. If the topology is enlarged to 5-layer, is there any changes that can be observed in terms of time and accuracy?

3. By using different value factors for mini-batches, what are the accuracy and time cost difference?

4. Compare the accuracy and time needed to compute when input features are changing, for example $64\times64\times3$, $128\times128\times3$, $256\times256\times3$ and $512\times512\times3$.

CHAPTER ③

Theory and Analysis: Face Recognition

This chapter elucidates how face recognition works and why it deserves attention. Face recognition and face verification are examples of methods for biometric authentication, biometric *means that physical information of an individual are measured to verify identity by unique shapes and structures [55]. Therefore, key differences between face detection, face recognition and face verification are explained. Relevant algorithms that may be used in the context of this project are presented and analysed. At the end of the chapter, several relevant case studies are proposed.*

3.1 FACE DETECTION

Face detection is anonymous by definition and particular algorithms normally discerns a *human face* by focusing upon distinct facial features to return a quantitative determination but stop at attempting to obtain a positive identification, hence there is no requirement for the digital image to be matched against a database of known identities [56]. The sole purpose of face detection is to determine if there is a human face in a given image. In other words, it is a procedure that distinguishes facial photos from non-facial photos. Several different algorithms can be used for this purpose, namely:

a. Histogram of Oriented Gradients (HOG)
b. Local Binary Patterns Histogram (LBPH)
c. Eigenfaces and Fisherfaces
d. Facial Landmarks

However, in this research, the focus will be on implementing LBPH as its mathematical approach is easier to understand and to implement. LBPH will be used as a pre-processing method to alter dataset before they are passed as an input to a neural network.

HOG, LBPH and facial landmarks can be well-established in real-time settings. Nowadays, they are utilized by social media, mobile phone camera as well as purpose-built digital cameras which have the capability to detect and certain device can also track facial movements. Faces that are identified are usually marked by a coloured square for user's convenience. If face detection precedes passing the picture to face recognition algorithms, the system makes sure that face recognition algorithms would only deal with input data where a face can be identified. Thus, non-facial photos that could lead to anomalous output are prevented from entering face recognition algorithms.

3.1.1 Local Binary Patterns (LBP)

The concepts behind LBP originated in a 1990's paper concerned with a new approach to texture analysis [57]. This work by Wang and He was extended, and LBPH as such was first formulated in 1994 by Ojala, Pietikainen and Harwood [57], [58]. LBPH soon became widely used for various tasks based on texture classification, and many versions of LBPH were developed over the years [59].

LBPH is not an extension from HOG but works somewhat similar to it and when combined, they can improve detection performance [60]. It is based on *Local Binary Patterns (LBP)* which labels the pixel of an image by thresholding the neighbourhood of each pixel and the outcome is a binary number [61] therefore this process reduces the data vector required for calculations significantly. LBP process produces one output; that LBP image and from this, a histogram to count frequencies of uniform bins in a given LBP image can be formed.

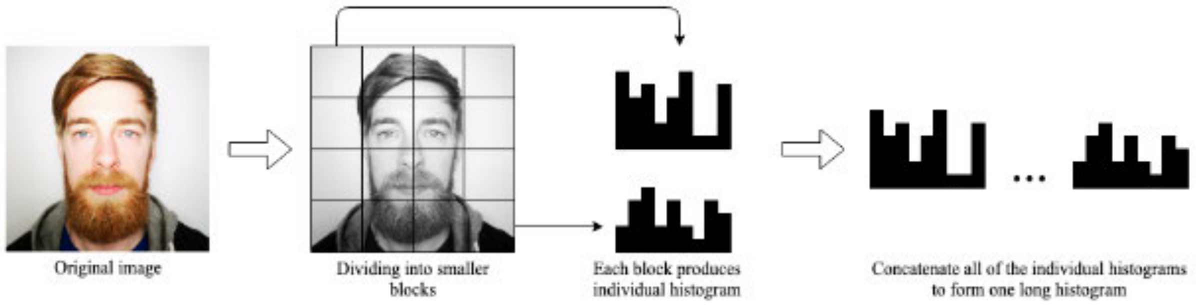
The process starts by converting the target image into grayscale. The image is divided into windows and the centre pixel, x, in the window is thresholded against its 8 neighbouring pixels $x_0 - x$, $x_1 - x$, $x_2 - x$, ..., $x_8 - x$.

x_4	x_3	x_2
x_5	x	x_1
x_6	x_7	x_8

Figure 28 : A window of 3×3.

In order to turn them into an 8-digit long binary number, values smaller than zero are converted to 0, and values equal to or higher than zero are converted to 1. This yields eight numbers per pixel that is then concatenated to form a binary number. It is then converted into a decimal number. The process is iterated for the rest of other windows. As the process ends up with an 8-digit binary number, it means there are $2^8 = 256$ different possible values for each window.

However, only 58 out of these 256 values are uniform, which means that in their binary forms there are at most 2 transitions from 0 to 1, or from 1 to 0. Bin number 59 will be assigned for all other non-uniform values. This is why the aforementioned histogram can be normalized by reducing the number of bins to $58 + 1 = 59$. After that, histograms from all windows are concatenated to form a final histogram. Then, a histogram can be plotted that visualize the occurrence of uniform bins in a given image. The facial image transformed into LBP image as well as the final histogram can be seen in Figure 30.



Original image
Dividing into smaller blocks
Each block produces individual histogram
Concatenate all of the individual histograms to form one long histogram

Figure 29 : Visualizing process of transforming an image to a histogram.

Due to the *local* nature of a histogram, if a change occurs in a small area of the image, the rest of the histogram will not be affected. Therefore, this technique can be used to identify changes in facial expressions.

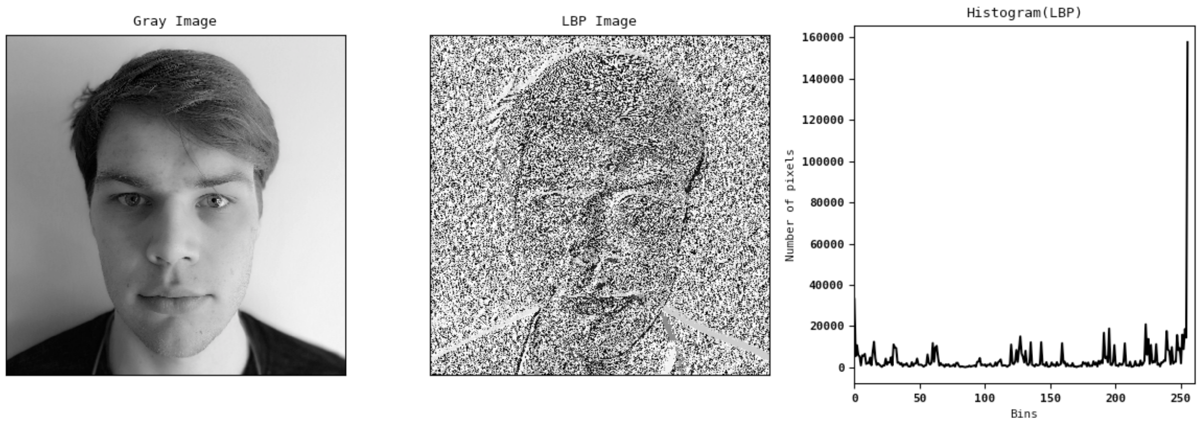


Figure 30 : Facial image transformed into LBP image (middle), their occurrence is plotted in the final LBP histogram (right).

LBPH can be a good candidate as it gave an output that may be considered unique due to its uniformity. Due to the nature of LBP image and LBPH which produce unique information, this research is interested in using this technique as dataset pre-processing.

3.2 FACE VERIFICATION

Although this project does not utilize face verification, its description is presented here in order to clarify terms. Face verification is 1×1 comparison and it decides if this given information matched, the output is a binary of *‘Yes’* or *‘No’*. An algorithm for face verification concentrates on making a positive identification of the individual against a database that archives personal information by thresholding a metric for confidence factor to avoid improper identification [56]. At first was developed for law enforcement usage, these days such technology has already been implemented commercially for many different purposes. Face recognition and face verification are becoming a standard for biometric applications. That is because it is relatively easy to put a face recognition or verification system into practice. Such technology is contactless and non-intrusive, unlike fingerprint identification; and it is relatively easy to obtain high quality and precise data in real-time.

The task for face verification is to compare two facial images and decide if there is the same person on both of them. In such a case, methods developed for comparison of two pictures are sufficient. They are based on one-to-one mapping; therefore, they can be faster, simpler, and require no training. This concept is being explained because it is related to face recognition which then can be related to the next sub-chapter, Face Recognition.

3.3 FACE RECOGNITION

Face recognition is an example of a more general problem area called *visual pattern recognition*. It concentrates on making a positive identification of an individual against a database that archives known information by thresholding a metric for confidence factor to avoid improper identification [62]. It first was developed for law enforcement usage, these days however, such technology has already been implemented commercially for many different purposes. It works by comparing an input image against a database of labelled facial images [9] in order to output an identity of the face in the new image [63]. Primarily, it uses landmarks such as the nose, mouth and eyes to identify a face. Shapes and proportions of these features are unique to each individual. These features are converted into a numerical template which is called a *faceprint*. The faceprint is compared against other faceprints in the database to find the closest match [55]. It means that face recognition methods must perform $1 \times N$ mapping, because there are many different labels in the database and only one of them has to be chosen as an output; but not necessarily so the image may not match anyone. According to the USA's *National Institute of Standards and Technology (NIST)*, the industry success identification rate is 99.7% [64].

There exist various algorithms that can be used to perform face recognition, and neural networks are one of them. A neural network can be trained with specific facial images from properly labelled dataset. Significant facial features of each person are learned by the neural network during the process of supervised training. Once the learning process is done, then the trained model can be used to identify the identity when a new unlabelled face image is being passed. If the person whom the face image belongs to exist in the dataset for training, then neural network will output the highest classification it has confidence with. If that is not the case and the confidence level below a certain threshold it can be programmed to produce an output *"Identity Unknown"*.

3.4 SUB-CONCLUSION

Detection, recognition and verification of faces have different purposes but they are somehow related when put into a linear process as recognition and verification can only work if detection returned a positive result of possible faces. Also, LBP can be a good candidate to study considering that is can produce two unique information in terms of LBP image and LBP histogram.

Based on the analysis for face recognition, a number of relevant case studies are:

1. Can the input for a neural network be pre-processed by LBP?

2. Can two outputs produce by LBP work for neural network training for face recognition?

3. By implementing face recognition using traditional algorithm, can the results be compared against results from neural network?

C H A P T E R

④

Theory and Analysis:

Dataset

This chapter will explore the last domain which is the dataset, another important domain for this research. Background regarding dataset will be explained and how it is being used. Later it will relate with the requirements for this project. At the end of this chapter, relationship between all three domains will be summarized.

Dataset is an important domain for this research as it is the source for data for training purposes. It is acts as a bridge between neural network and the classification process, also for the purpose of this research: face recognition. A face recognition dataset needs faces to work and amassing enough images can be both a logistical and an ethical minefield. Therefore, a scope for logistical requirement must be explained and the ethical as well as legal requirement to be adhered.

4.1 DATASET EXAMPLE

There are large repositories online for dataset for example Google Dataset Search[1], Kaggle[2], UCI Machine Learning Repository[3], VisualData[4], 3D Human Pose Estimation[5], etc. Datasets served for different purposes for example population dietary, finance, search engine keywords trend, human genome or sentiment analysis. Also, there are a number of datasets for face recognition namely *Labelled Faces in the Wild*[6] which contains $13,000$ labelled images of human faces; *The Yale Face Database B*[7] which contains $5,760$ single light source images of 10 subjects each seen under 576 viewing conditions; however, a closely relevant to this project

[1] https://toolbox.google.com/datasetsearch
[2] https://www.kaggle.com
[3] http://mlr.cs.umass.edu/ml/
[4] https://www.visualdata.io/
[5] https://medium.com/neurohive-computer-vision/new-datasets-for-3d-human-pose-estimation-45cd320e37bd
[6] http://vis-www.cs.umass.edu/lfw/
[7] http://vision.ucsd.edu/content/extended-yale-face-database-b-b

is *Face Recognition Data*[8] from University of Essex, UK which contains image of 395 individuals (male and female) with 20 images per individual of various racial origins and some individuals are wearing glasses and beards.

4.2 BUILDING A DATASET

As this research is aimed at implementing neural network for face recognition, relevant dataset is of course the one that has faces. Despite the number of different dataset available on the internet for free download, most of them fell short in terms of sample number per person and also the photo dimension which either too small or compressed which means that they lost a lot of information. As the purpose is to implement the knowledge in neural network and with the high performance computer availability for this research, a decision is made to push the boundary to the maximum by building a larger dataset with larger dimension of images and more importantly with different poses and light illuminations to press towards the non-traditional face recognition which is based on neutral or forward looking faces [65].

4.3 SPLITTING DATASET

If dataset is too large, it must be split up to lower the processing time. However, as most dataset is quite small, all the features are normally taken. But consideration must also be based on the model of chosen neural network. For example, Artificial Neural Network (ANN) can be built for as small as 3-layer network and therefore a large features from a dataset normally can be fit in into memory easily but it will be difficult and time consuming when Convolutional Neural

[8] http://cswww.essex.ac.uk/mv/allfaces/index.html

Network (CNN) is considered as it has an additional layer which work on individual images, where ANN can just form a large matrix for a one time calculation.

Typically, a dataset is split into three parts: training, validation and testing. 40% training, 40% for validation and 20% test but these ratios are not a fixed number [54], it must be based on the requirement of a neural network [66]. From statistical point of view, if a same weight can be applied through testing, the result should have the same weight as training. It meant that, the training and testing should have been separated equally. The difference of magnitude when splitting can be projected by using this example:

A dataset has a total of 100 data features, 80 is used for training and 20 is for testing. If the accuracy rate given as $\gamma_{training}$, and testing result was expected to emulate similar result, $\gamma_{testing}$. However, the magnitude of accuracy from both γ*s* are from two different weights. As testing only represent 25% of training. In order for both γ*s* to be equally weighted, they should have been using similar weight that is 50 for training and 50 for testing for a fair chance for the images to compete in the network and also for fair change of statistical comparison.

4.4 DATASET FORMAT

Dataset format must be based on the requirement of a particular neural network. Neural network models often than not can't be trained on using raw pixel values, that is using the original pixel values which is in the range of 0 to 255. The reason is that the network uses a weighted sum of inputs, and for the network to both be stable and train effectively, weights should be kept small to avoid fluctuations and higher variations. Therefore, images pixel values must be scaled using certain scaling method prior to training. Three main approaches to scaling pixel values are normalization, whitening (also known as centering) and standardization.

4.4.1 Normalization

Pixel values are scaled to the range between 0 and 1. Normalization is often used as it is always safe to assume that pixel values are always in the range between 0 and 255, so the normalization can be done by dividing each pixel by 255. However, a proper normalization should check what is the maximum pixel value in an image instead of assuming that it is 255. Also, some dataset may need to be rescaled, for example images are normally have a rescaling process called normalization to reduce RGB matrix colour which ranges between 0 to 255 to new values between 0 and 1. The purpose of rescaling is to train a neural network faster and to increase accuracy as values are not fluctuating between big numbers [67]. Also by normalizing data between 0 and 1 will improve activation function is sigmoid or hyperbolic tangents are used as both are known to have vanishing gradient problem when values are bigger than 5 or smaller than -5.

4.4.2 Centering

The mean pixel value is subtracted from each pixel value resulting in a distribution of pixel values centred on a mean of zero. Centering is often promoted as the preferred approach as it was used in many popular papers, although the mean can be calculated per image (global) or channel (local) and across the batch of images or the entire training dataset, and often the procedure described in a paper does not specify exactly which variation was used.

4.4.3 Standardization Features

The pixel values are scaled to a standard Gaussian with a mean of zero and a standard deviation of one. The choice of scaling is not a fixed process and all available choices should be done before one or two scaling method are chosen for the experimentation. However, not all the time that a long vector can be fed into a network, CNN for example works on the original image and therefore requires channel reshape.

$$X_{std} = \frac{X - \mu}{\sigma}$$ *Equation 21*

μ is the mean of an input feature X and σ is the standard deviation for each input.

4.5 DATASET FOR PROJECT

Finding a proper dataset is a challenge too and most datasets came from previous research for a particular interest and possibility to be extended is almost null. Therefore, this research may need to build its own dataset, which gives an advantage in terms of expanding the image, pose and facial expression, angles, lighting shades and background ambient. Besides that, as this project is aiming to train neural network through forward and backpropagation iteratively on each training data and each data will reflect the learning ability, therefore the size of dataset for this project must be at optimum so that the network can have enough variety and better at predicting outputs [14].

4.5.1 Project Dataset

As this project has been defined for face recognition, it is clear that requirement for dataset revolves around acquiring facial images. Therefore, in order to build a proper dataset, a decision has been made to collect facial photos from 100 volunteers with each one will help to provide 220 photos during 10 different sessions. If the target is accomplished, the dataset will contain 22,000 facial images. These images will be separated equally for training (10,000 facial images) and testing (10,000 facial images) and additionally 2 from each session for verification (2,000 facial images). Even though that the training is expected to contain 10,000 facial images and by certain remarks, it is considered a lot but it is by comparison very small when compared to a testing by IBM for its *IBM i2 Facial Recognition* [68] which uses 1 million facial images for its training dataset to truly fight bias in face recognition [69].

4.5.2 Consent

As collecting and storing facial images are considered an encroachment into personal privacy space, hence a proper management is required so that this experiment is operating within a proper legal framework. In order to be able to use facial images for the experiment, a written consent must be acquired from each volunteer. Each consented volunteer will be provided a unique number for the purpose of organizing their facial images for the experiment and when writing this report as to comply with the requirements for privacy and anonymity law [70].

A mixture of volunteers with different facial structures are expected to contribute towards the dataset: without/with prescription glasses, different make-up, different hair (or bald, length, colour), different skin tone, with/without facial hair (beard/moustache). As the interval between sessions is somewhat too short, it should be argued that wearing a hat, jacket, shirt or scarf that covers around the neck should also impact the ability for face recognition. Also, men mostly have noticeable changes in the form of facial hair, especially around the chin, cheeks, and upper lip region within few days. Volunteers are also asked to provide their gender and age-group in the consent form.

4.5.3 Settings and Facial Expressions

In a standard face recognition, to achieve high accuracy, images must be taken in well-illuminated environment and clear facial details. This is also the requirement set up by the *International Civil Aviation Organization (ICAO)* on passport image which requires an even lighting, plain background, neutral facial expression and individual to face toward the camera [71].

But, face recognition should be able to work in another environment outside of its ideal as there will be different variables that can impair the accuracy for example variation in pose, illumination, resolution, occlusion, etc [61]. A research from *Carnegie Mellon University* found that by applying correlation filters, a neural network can be trained to improve accuracy for face recognition when faces are obscured, poorly lit or facing away from the camera [72].

One of the methods to expand the size of training dataset artificially is by deploying data augmentation by creating modified versions using images from the dataset. Neural network training process on more data can be expected to improve the accuracy of a trained model and the augmentation techniques can create variations of the images that will be good for the training [73]. The intent is to expand the training dataset with plausible examples of different variations and the augmented data are only used for training. But the case is valid for a dataset acquired from a third party as it cannot be expanded using additional observation. However, for this project, *data augmentation* can take place in the form of face's muscles movement and movement of face at different angles.

During the photography, volunteers will be asked to stand against a neutral coloured background with camera static at one place. Volunteer then will move his/her face around while making facial expressions as to consider poses other than frontal. Lightings will be from natural lights and as the condition in Denmark where the weather varies, therefore works as an additional factor to be benefited from as lights are shades are introduced naturally. During each session, volunteers are asked to show different facial expressions: neutral, smiling, winking, happy-face, sad-face; face movement in different angles: forward-looking, downward, upward, leftward, rightward and other different angles but facial expressions can still be seen and captured.

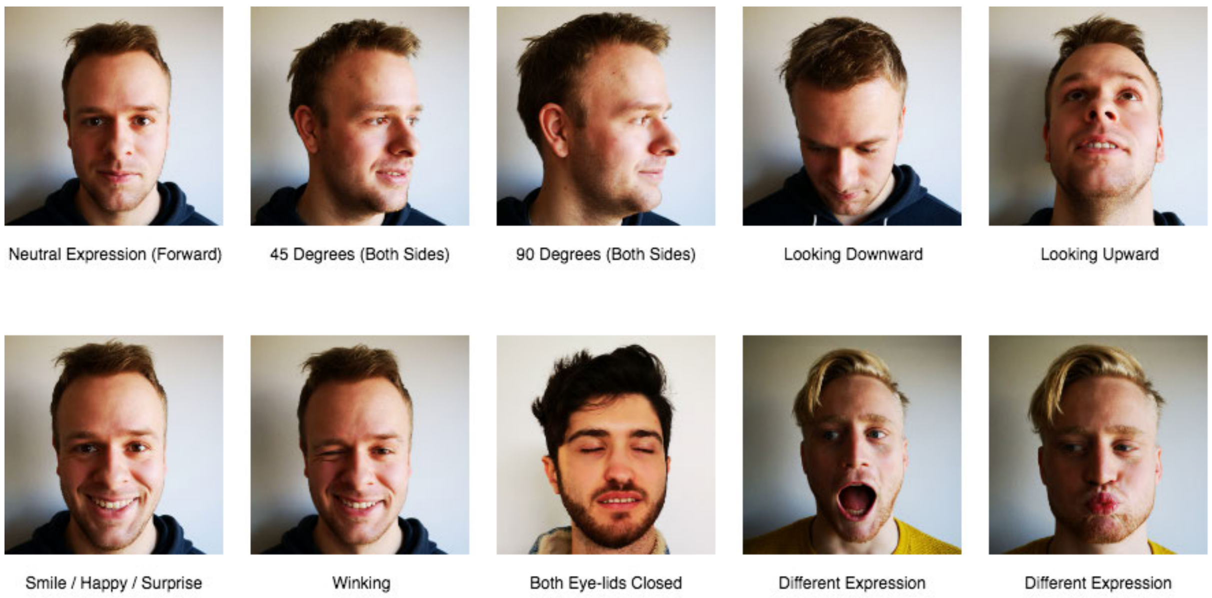


Figure 31 : Facial expression expected from volunteers. First row is neutral looking in five different positions and the second row is miscellaneous expressions in order to capture different facial muscles movement.

4.5.4 Editing Dataset

All images will be edited by cropping using a square ratio to get a $1:1$ dimension and then resized to $1200 \times 1200 \times 3$ pixels. The purpose of cropping is to get an equal size for width and height which will be equally positioned when converted to a long array when being used as input features. This dimension is expected to keep the quality of face images optimally without using too much space.

4.5.5 Compiling Dataset

A number of dimension will be considered from the lowest 64×64, 128×128, 256×256 up to 512×512 pixels. Facebook reported that their research team have successfully used a dimension of 64×64 for one of its neural network training [74].

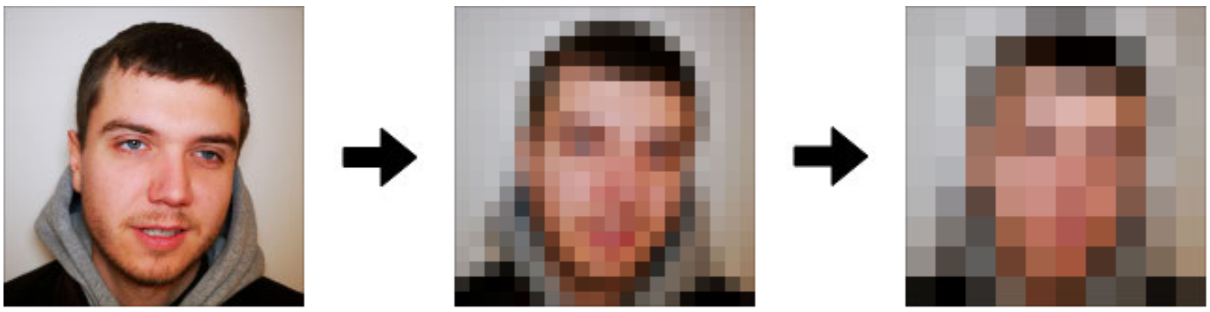


Figure 32 : Effect examples of when an image's dimension is reduced by smaller factors, causing pixilation and clearly every reduction increases the losses.

Method 1 : Pickle Array

For the purpose of experiment which will include different image dimension, images will be resized and saved into Python Pickle files for quicker loading. Images will be resized and saved into separate files based on the dimension. Images will be resized to 64×64 and 128×128.

Each image will be converted to a long array, therefore a 64×64 with 3-colour depth will be an array of $12,288$. Each image will be prepended with two additional values: the first one is for indexing and the second for the volunteer's identification label, therefore extending the array to $12,290$ values. The purpose of indexing is to solve a problem when converting expected label during one-hot as it requires indexing to be within the range of classification number, while volunteer's identification label is a fixed number and during finalizing dataset, some volunteers may be dropped out for various reasons and will create a problem when passed to one-hot function.

When images have been turned into their individual arrays, they will then have stacked vertically to form a dataset matrix. For example, 500 images with $64 \times 64 \times 3$ when merged together with its respective label will form a matrix with the dimension of $12,290 \times 500$.

Method 2 : Original Images

Consideration must also be made for the dataset files. As the images grow and converting to arrays will lead that images are losing compression and therefore increasing the size, different methodology to organize the dataset must be considered. Large file considered in *Method 1* will produce a very large dataset files and when loaded will create time complexity and overloading the memory. Therefore, another method must consider that is by having images organized into folders and loaded when necessary. However, to reduce the overhead time to read the images name into a list, the list must be generated and stored for later access.

4.6 SUB-CONCLUSION

A dataset with label can help to increase face recognition rate, but the downside comes when a dataset is quite large as typically datasets can be very large and contain many facial photos of many different people which will create a time overhead during each processing. Here comes the part where neural network can play its role to reduce processing time and also to avoid misclassifications by turning each photo into a faceprint. Because of the amount of data, a really high number of computational operations would be required by traditional methods to yield an output. Because of their non-linear nature that tackles well problems concerning large amounts of data, deep neural networks are typically used for applications of face recognition.

Alteration on pose or angle will alter mathematical values greatly and therefore altering the ability for a computer to recognize. Not only pose, but the skin tone colour also affecting result of recognition as research has shown that facial recognition is biased towards a Caucasian male with a 99% success rate and the error arise as the skin colour changes with women with darker skin has only 35% success rate [75]. But, if there are more than one photo of a person from a different angle and different shades can be trained using a particular neural network, chances for a computer to recognize a person can be increased dramatically.

There are a number of case studies that can be explored for this chapter. Some of the ideas are:

1. Training with images in RGB versus grayscale to compare the speed and accuracy. It must be noted that RGB matrix is three times larger than that of grayscale. Can an assumption be made that the speed will be a third to RGB's but accuracy should not be affected as important features remain in the same position?

2. If images are separated into three colour channels and only one channel is chosen for training and the results are then compared for full channels' training, what are the accuracy loss and speed increment?

3. If images are resized for 64×64, 128×128 and 256×256 for training, what are the accuracy and time cost difference?

C H A P T E R

⑤

Design and Implementation

Before an experiment can take place, a design methodology should be considered as guidelines during the programming. This chapter will explore and detail a plan on how to properly implement a design pattern in terms of coding so that a standard programming method can be followed throughout the whole process. Different software architecture models and design methodologies exists to assist with software design, however for this specific purpose the choice has fallen on MVC (Model-View-Controller) and UML (Unified Modelling Language).

5.1 THE MODEL—VIEW—CONTROLLER (MVC)

The MVC is an architectural design that separates presentation and interaction from the system data. The system is divided into three logical components that interact with each other. The *Model* manages data and associated operations on the data. The *View* defines how the data is presented to the user. The *Controller* manages user interaction [76]. Ideally the user only interacts with the software via the *View* component that should then pass the user input to the controller. The *Controller* may communicate with itself, as well as the *Model*. The *Controller* then passes a result back to the *View* [77].

The MVC approached implemented in this coding, were separated into three components; the first one for Controller where a set of command for training, saving into external files or plotting graph from saved output is being invoked; second is Model where it received configurations and input from Controller; and the third one is View which received output from Model and saved them into external files (text, pickle or image) for review.

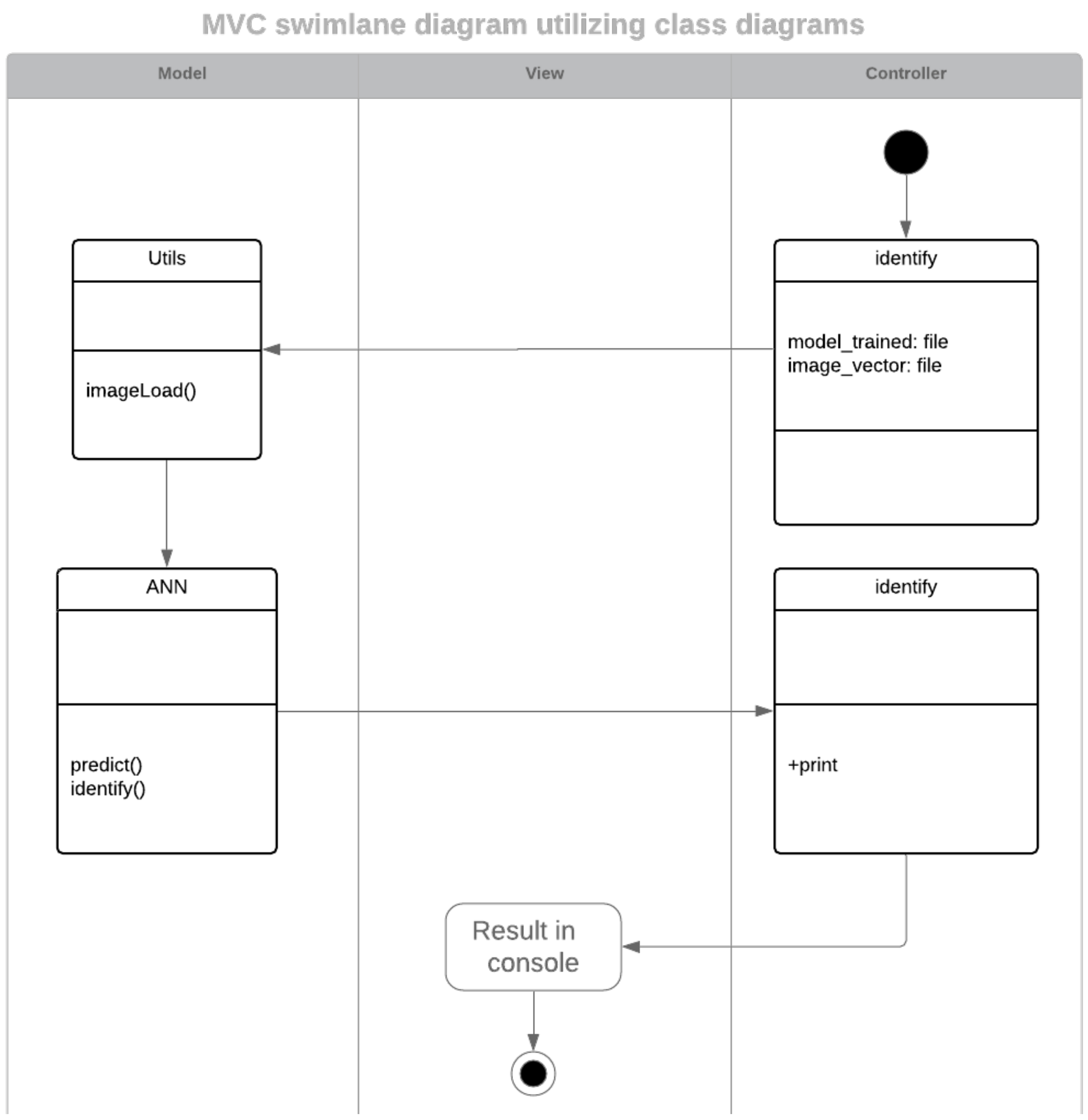


Figure 33 : The user selects a trained model and an image to check against it. Next the image is loaded, after which it is compared against the "database" in the trained model for identification. The result is passed back to the controller that prints it to the console.

As can be seen from Figure 32 the software does not follow the MVC guidelines precisely as the user interacts directly with the *Controller*. This can be changed by changing the code in the *identify* class to prompt the user for the location and name of the model as well as image file, or by implementing a graphical user interface so that the user can visually pick a model and navigate to an image file to verify against it. However, time constraints meant that it was not possible to implement for this version of the software, neither was it deemed a priority as the only users of the software will be limited to the authors of this report.

5.2 UNIFIED MODELLING LANGUAGE (UML)

The Unified Modelling Language (UML) is a set of 14 different diagram types used to model software systems. It came into existence in the 1990s where several similar object oriented notations were merged to create UML [76][78]. It is important to note that a system model is not intended as a complete representation of the system. A model purposely leaves out leaves out details in order to make the model easier to understand. Likewise, not all attributes and methods are represented in the class diagrams for this system, neither will the diagrams cover 3rd party libraries utilized in the code.

The Model diagram is not part of the official 14 UML diagrams, but has been used as it provides a good overview of the different packages and classes in the system and of their dependencies [78] and can be seen in Figure 34.

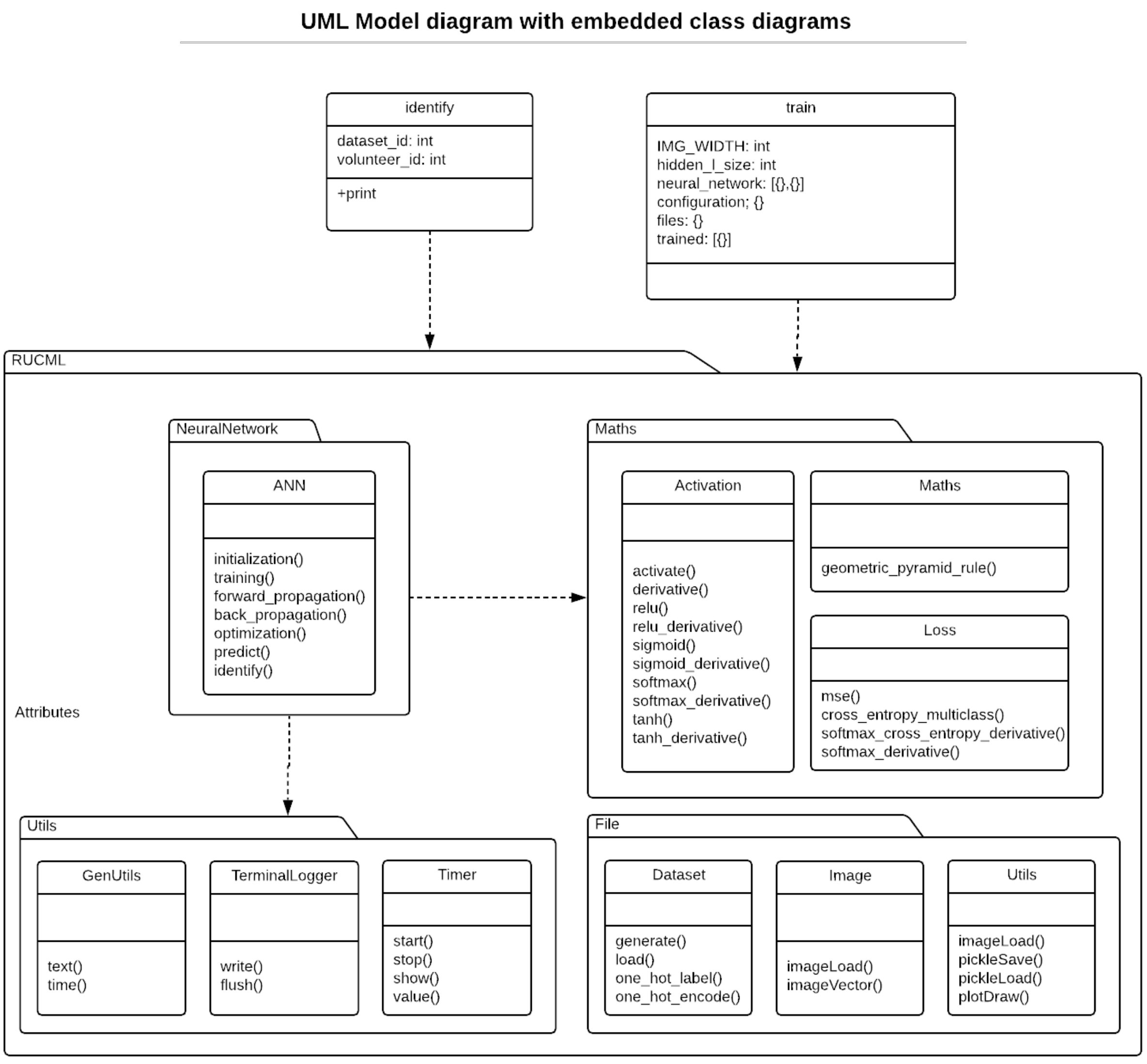


Figure 34 : The model diagram.

5.3 IMPLEMENTATION

This process involves the mathematical concepts that have been explored in chapter 2, as well as chapter 3 and 4 to come out with a Python code that is expected to *learn* and *recognize* face from the dataset.

5.3.1 Object Oriented Programming

Object-oriented programming (OOP) is a concept that establishes existence of objects which can contain data and procedures. Thanks to that, code written under guidelines of OOP is more readable and less prone to bugs, that is because of improved organization of code. In the code written for this research, OOP ideas are implemented as separation of code into modules and classes [79]. This way it was possible to separate the core of the artificial neural network from code responsible for image processing and the rest, into different files.

For most of the program, modules were used for code segregation, only in few instances were used classes. Classes benefit from the ability to be instantiated multiple times as several different objects, whereas modules can only be imported once.

5.3.2 Implementation of a Neural Network

The module named `ANN` contains code for the artificial neural network and can be found in `RUCML/NeuralNetwork`. Three core processes that provide basic functionality are forward propagation, backpropagation, optimization. Each of them has a respective method. Basic components like layers, weights and biases are in a form of variables.

Simple mathematical functions that are necessary for the functionality of forward propagation, backpropagation and optimization are all to be found in a package named `Maths`. Depending of application of these functions, they are located in modules `Activation`, `Loss`, `Maths`. An example of code for these functions can be seen in Figure 34.

```
def ReLU(sum):
    activated = np.maximum(0, sum)
    gc.collect()
    return activated


def ReLU_Derivative(activated):
    derivation = np.array(activated, copy=True)
    derivation[activated <= 0] = 0;
    gc.collect()
    return derivation
```

Figure 35 : An example of code for an activation function and its derivative.

A. Layers

Layer sizes and other hyperparameters are configured when the module ANN is called, not inside the module itself. Specifications of layers are expressed in the argument **`neural_network`** and other hyperparameters in the argument **`configuration`**. This is illustrated in figure 1.

```
Training(
    neural_network,
    X,
    Y_expected,
    Y_hot,
    configuration,
    files,
    X_testing,
    Y_testing
)
```

Figure 36 : Necessary arguments to train the neural network.

B. Weights and Biases

Initial values for weights and biases in the model are generated by a loop through all layers of the neural network. During initialization, weights are randomly generated using standard distribution between $(0, 1]$. All that happens in accordance with size specifications of the layers, as it can be seen in figure 1. Weights and biases are stored in the python dictionary named

`cache_parameters` with keys for example **`W1`** for weight connecting to layer 1 and **`b1`**, bias for layer 1. The number will represent the subsequent hidden layer.

```
for index, layer in enumerate(NN):
    layer_index = index + 1
    layer_input_size, layer_output_size = layer["dimension_in"], layer["dimension_out"]

    cache_parameters['W' + str(layer_index)] = np.random.normal(loc=0.0,
                                                                scale=np.sqrt(6) / np.sqrt(
                                                                    layer_input_size + layer_output_size),
                                                                size=(layer_input_size, layer_output_size))

    cache_parameters['b' + str(layer_index)] = np.ones((layer_output_size, 1))
```

Figure 37 : Initialization of weights and biases in the model.

C. Forward Propagation

Forward propagation is implemented as a loop through all layers. Vectors representing results of a summation function and of an activation function are stored in the python dictionary **`cache_calculations`** with keys for example **`Z1`** and **`A1`** for sum and activated values for hidden layer 1. This can be seen in figure 1.

```
for i in range(1, int(len(NN) + 1)):
    cache_calculations["Z" + str(i)] = np.dot(cache_parameters["W" + str(i)].T,
                                              cache_calculations["A" + str(i - 1)]) + cache_parameters["b" + str(i)]
    cache_calculations["A" + str(i)] = MathsActivation.Activate(cache_calculations["Z" + str(i)],
                                                                NN[i - 1]["activation"])
```

Figure 38 : Implementation of forward propagation.

D. Backpropagation

Backpropagation is implemented as two for-loops that loop backwards through all layers of the neural network. The first loop implements the delta rule. The second loop calculates gradients that are stored in the python dictionary **`cache_gradients`**, their keys are for example **`dW1`** for weights gradients, or **`db1`** biases gradients for hidden layer 1. See figure 3 for the code.

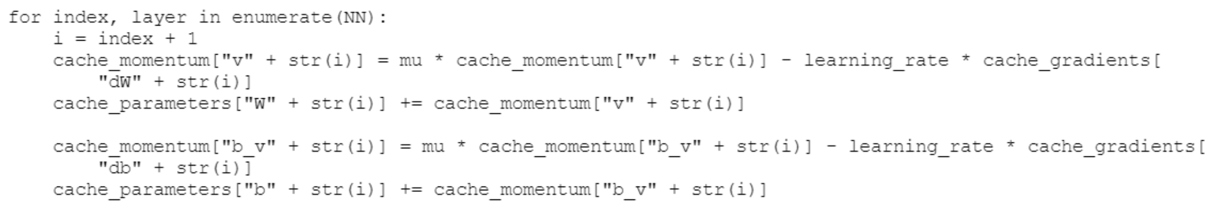
```
for i in range(len(NN) - 1, 0, -1):
    ## delta = dL/dA * dA/dS
    delta[str(i)] = (cache_parameters["W" + str(i + 1)].dot(delta[str(i + 1)])) *
        MathsActivation.Derivative(cache_calculations["Z" + str(i)],
        NN[i - 1]["activation"])

for i in range(len(NN), 0, -1):
    dL_dW[str(i)] = delta[str(i)].dot(cache_calculations["A" + str(i - 1)].T)
    cache_gradients["dW" + str(i)] = dL_dW[str(i)].T / m +
        (regularization_constant / m) *
        cache_parameters["W" + str(i)]

    cache_gradients["db" + str(i)] = np.sum(dL_dW[str(i)], axis=1, keepdims=True) / m +
        (regularization_constant / m) * cache_parameters["b" + str(i)]
```

Figure 39 : Implementation of backpropagation.

E. Optimization

After each iteration, weights and biases are updated. This is done by looping through all layers of the neural network to calculate momenta and update weights and biases, as figure 4 illustrates. Momenta are calculated from gradients (**`cache_gradients`**) and stored in **`cache_momentum`**. Weights and biases are updated with the momenta and new values are stored in **`cache_parameters`**.

Two variables that regulate optimization are **`mu`** (coefficient of momentum) and **`learning_rate`**.



```
for index, layer in enumerate(NN):
    i = index + 1
    cache_momentum["v" + str(i)] = mu * cache_momentum["v" + str(i)] - learning_rate * cache_gradients[
        "dW" + str(i)]
    cache_parameters["W" + str(i)] += cache_momentum["v" + str(i)]

    cache_momentum["b_v" + str(i)] = mu * cache_momentum["b_v" + str(i)] - learning_rate * cache_gradients[
        "db" + str(i)]
    cache_parameters["b" + str(i)] += cache_momentum["b_v" + str(i)]
```

Figure 40 : Implementation of optimization with momentum

5.3.3 Memory Management

Global variable caching was deployed using dictionaries. This solution was implemented as a reaction to a problem that arises when a large number of variables is created during the many computations that are necessary for a successful training of a neural network. Initially, many local values were passed from method to method which generated a lot of variables that needed to be stored in memory. Furthermore, generating new variables all the time wasted computational power and increased total time necessary for the program to run. Dictionaries are a good way to combat this as they are global, i.e. they can be accessed by any method in the program so that a value can be stored without the need for a new variable. Memory- and time-wise, this improved performance vastly. Names of dictionaries used in the program are **`cache_parameters`**, **`cache_calculations`**, **`cache_gradients`**, **`cache_momentum`**.

5.3.4 Processing of Output

This research is based on classifying facial images to their respective volunteers. Therefore the classification is made by using Softmax activation function in the output layer to squashed the incoming signals into values between $[0, 1]$ and distribute the number of probability into the group of classified label. Softmax produces probability distribution for each class and as the distribution sum for each class equals to 1, then each distribution can be defined in terms of percentage. This percentage can also be defined as the confidence level.

```
if len([*filter(lambda x: x >= 0, Y_hat)]) > 0:
   for i in range(0, X.shape[1]):
      prediction[0, i] = np.argmax(Y_hat[:, i])
```

Figure 41 : Implementation to get predicted label

In the code snippet above filter a list to make sure that the output is not empty and then for each input features in the *i*-th position, will be returned the maximum value from each respective *i*-th column from the output layer. The reason to use this for-loop because there can be more than one input column, but during identification, most of the time, there's only one input image. Therefore for-loop will stop after 1 iteration. The prediction will return the position of a particular class in the one-hot binary encoding.

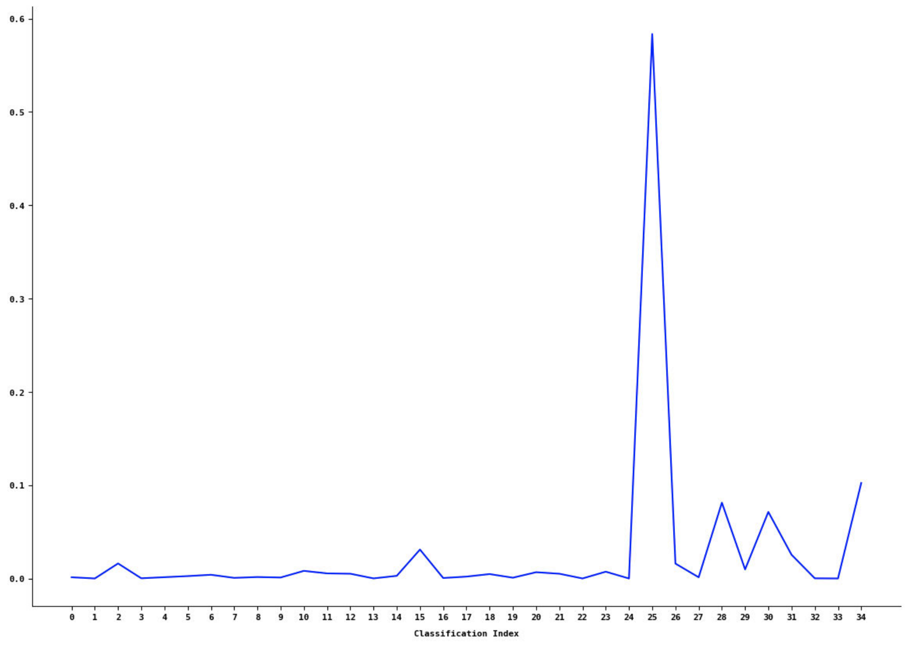


Figure 42 : An example of probability distribution for a prediction with 35 classes. It can be seen that the peak is 25, which points to the one-hot binary encoding index value.

The value from **`Y_hat`** is probability distribution and can be extracted using,

```
confidence = np.amax(Y_hat)
```

Figure 43 : Function to get the highest confidence level

It returns the biggest value from **`Y_hat`** array.

5.4 SUB-CONCLUSION

Considering the level of proficiency in programming software, or the lack thereof. More time should have probably been used initially for the design of the software. It has been incredibly useful though to have the MVC model in mind when developing the software as it provided a good way of separating the code into smaller bits/problems.

The UML charts were reworked several times as the code progressed, showing the need for a more structured design phase. The artificial network has been implemented successfully though, despite the shortcomings of the design phase, and a lot of learning has been derived from the process.

CHAPTER

⑥

Case Studies and Experiments

From the theory and analysis in chapter 2, 3 and 4; suggested case studies will be chosen for implementation and have their results recorded in this chapter. Full observations will be discussed in depth in chapter 7. Four different case studies have been chosen and the training and testing process will be queued on High Performance Computer provided by Department of Science and Environment.

6.1 SELECTED CASE STUDIES

Similar dataset is used for all of the experiment, with 100 images are for training and another 100 for testing. It has been argued in chapter 4 regarding the choice of similar number for training and testing. Experiments are made by changing the size of the hidden layer to observe the time complexities, cross entropy loss as well as accuracy for training and testing.

The case studies are:

a. dataset with colour images,
b. dataset with grayscale images,
c. dataset with LBP images, and
d. dataset with LBP histograms.

Different hidden layer sizes *(L1-size)* will be experimented with. They are 256, 512, 768, 1024, and additional two more will be based on Geometric Pyramid Rule (GPR). The dataset image dimension is resized to 64×64 and 128×128. These dimensions are chosen because 256×256 images are too big and they increase time complexity for calculations on the server. While 64×64 provides a valid comparison for 128×128 in terms of data loss and how that would affect training and testing results. The positions of each input feature in the dataset are randomized before each training and testing but the same position placement remains the same through the epochs.

The neural network is set to have three layers; one for input, L_i; one hidden layer, L_1; one for output, L_o. The number of neurons for the input layer is determined by the number of pixels or the length of a histogram. The number for output is determined by the number of volunteers to be classified and the number for hidden are changed based on the discussion above. The activation function will be Sigmoid for hidden layer and Softmax for output layer.

A number of other variables were fixed based on trial and error during the implementation. The learning, α was chosen as 0.075 because during the implementation, a range of value was tested in the range 10^{-4} and 1 which resulted in a number of observations. For example when chosen a number too small, the process converge rather slowly through the epoch but produces a smoother graph, but is on the opposite side when chosen value close to 1 as the loss values were noisy and shows inconsistencies as the epochs continue, despite converging towards the smaller loss value. When tested between 0.05 and 0.075, it was observed that the loss graphs are showing more promising results, in terms of smoother gradient, less noise and converging between 200 to 400 epochs. The summary for fixed variables used for all of the experiment can be seen in Table 3.

Neural network architecture	3-Layer
Number of epochs	2,000
Learning rate, α	0.075

Total dataset for training	3,200
Total classification	32

Table 3 : Fixed variables to be used for all of the experiments.

Following the MVC approach from chapter 5, each experiment will invoke training separately in the *Controller* part. Each training uses similar *Model* and for each experiment, the *View* (in this case the output) will be saved into text, pickle and image file for analysis.

6.2 DATASET WITH COLOUR IMAGES

This experiment will be using original colour images that have been resized based on the previous discussion in the section 6.0.

6.2.1 Variables

Number of input features (64×64×3)	12,288
Number of input features (128×128×3)	49,152

Table 4 : Variables used in this experiment.

6.2.2 Results

L1-Size	Image Dimension	Training Time	Loss	Accuracy (Training) %	Accuracy (Testing) %
256	64	1:26:25	0.0503	99.97	93.75
	128	4:05:51	0.0451	99.94	93.56
512	64	2:34:59	0.0330	100	93.63
	128	5:21:24	0.0223	100	94.50
768	64	3:20:47	0.0268	100	93.84
	128	6:25:41	0.0165	100	94.16
1,024	64	3:55:40	0.0266	100	93.34
	128	8:00:16	0.0165	100	94.22
2,048	64	7:55:22	0.0246	100	93.41
	128	9:59:57	0.0153	100	94.16
GPR (110)	64	2:21:31	0.0627	99.97	93.59
GPR (221)	128	8:54:34	1.1193	76.56	66.97

Table 5 : Observations from 12 different experiments.

6.3 DATASET WITH GRAYSCALE IMAGES

In this experiment, each image in the dataset are transformed into a grayscale which has a single colour channel and also changed in terms of dimensions. The input feature length is then 3 times smaller when compared to colour images because grayscale only has 1 colour channel.

6.3.1 Variables

Input feature length (64×64)	4,096
Input feature length (128×128)	16,384

Table 6 : Variables used in this experiment.

6.3.2 Results

L1-Size	Image Dimension	Training Time	Loss	Accuracy (Training) %	Accuracy (Testing) %
256	64	0:58:16	0.1164	99.34	90.28
	128	2:21:33	0.0492	99.97	91.47
512	64	1:13:29	0.1062	99.44	90.19
	128	2:58:41	0.0349	100	91.78
768	64	1:24:25	0.0963	99.56	90.16
	128	3:44:49	0.0322	100	91.44
1,024	64	1:32:13	0.0929	99.56	90.03
	128	4:01:54	0.0300	100	91.28
2,048	64	2:16:24	0.0845	99.72	89.78
	128	5:55:40	0.0303	100	91.22
GPR (64)	64	0:43:44	0.1939	98.41	89.22
GPR (128)	128	3:23:49	0.0875	99.94	90.59

Table 7 : Observations from 12 different experiments.

6.4 DATASET WITH LBP IMAGES

The size of the window was tested by dividing an image into 2×2, 4×4, 8×8, 16×16 and 32×32. The fourth one 16×16 has been chosen to be used for the dataset transformation because the first three did not yield enough information in terms of independent input features. But 32×32 not only provides too much information but also quite noisy because the dimension for the dataset images are 64×64 and 128×128.

6.4.1 Variables

Input feature length (64×64)	4,096
Input feature length (128×128)	16,384

Table 8 : Variables used in this experiment.

6.4.2 Results

L1-Size	Image Dimension	Training Time	Loss	Accuracy (Training) %	Accuracy (Testing) %
256	64×64	1:32:51	0.0305	100	82.09
	128×128	2:35:42	0.0151	100	81.22
512	64×64	1:58:33	0.0258	100	81.66
	128×128	3:04:45	0.0112	100	80.72
768	64×64	2:27:05	0.0239	100	81.53
	128×128	3:08:26	0.0099	100	80.97

1,024	64×64	2:44:16	0.0226	100	81.94
	128×128	3:45:24	0.0092	100	80.38
2,048	64×64	3:49:06	0.0209	100	82.06
	128×128	5:08:51	0.0094	100	81.16
GPR (64)	64×64	1:12:52	0.0589	100	79.56
GPR (128)	128×128	3:23:20	0.0257	100	80.06

Table 9 : Observations from 12 different experiments.

6.5 DATASET WITH LBP HISTOGRAMS

Input feature length is 3 times smaller when compared to colour images because LBP image only has 1 colour channel. On top of that, more information is lost due to reducing the LBP images to histograms; therefore, ANN will be learning about the number of pixel occurrences from the histogram.

Size of the window is 16×16 as determined in the sub-chapter 6.3. The division into 16×16 windows produces 256 histograms and each histogram yields 59 bins. Concatenation of histograms produces 256 × 59 = 15,104 input features regardless of image dimension.

6.5.1 Fixed Variables

Input feature length (For image 64×64)	15,104
Input feature length (For image 128×128)	15,104

Table 10 : Variables used in this experiment.

6.5.2 Results

L1-Size	Image Dimension	Training Time	Loss	Accuracy (Training) %	Accuracy (Testing) %
256	64×64	2:25:58	3.4496	53.63	47.25
	128×128	2:37:19	3.0286	51.00	49.50
512	64×64	3:30:42	3.4484	58.03	51.06
	128×128	3:31:50	3.0357	52.38	50.56
768	64×64	4:25:06	3.4481	57.81	51.34
	128×128	4:23:05	3.0209	52.81	50.75
1,024	64×64	4:39:01	3.4472	59.59	52.97
	128×128	4:36:08	3.0175	53.59	51.38
2,048	64×64	6:14:59	3.4458	56.88	49.84
	128×128	6:12:19	2.9970	53.84	51.44
GPR (123)	64×64	1:18:51	3.4519	51.97	45.22
GPR (123)	128×128	1:18:52	3.0458	47.50	44.03

Table 11 : Observations from 12 different experiments.

6.6 SUB-CONCLUSION

The case studies with all twelve experiments in each one shows some promising results and it is the first stage to prove that the implementation for neural network is working and at the same time proving that face recognition can be achieved through several methods using neural network.

All four case studies presented in this chapter are comparable due to several fixed variables. Results are presented in numerical will provide a standardized reference ground for discussions that will follow in the next chapter. For readability, figures for all case studies can be found in the appendix.

CHAPTER

⑦

Discussion and Conclusion

This chapter will discuss, conclude and will give suggestions for future work with regards to neural network and also face recognition. It will begin with discussing the results obtained from experiments and recorded in chapter 6. Then followed by conclusion based on the observations before summarizing the whole project.

7.1 INTERPRETATION OF CASE STUDIES

All of the experiments in each case study have their variables fixed and the Artificial Neural Network (ANN) was only using a single hidden layer. Each experiment took place by changing the number of neurons in the hidden layer; $256, 512, 768, 1024$ and 2048. These numbers were chosen based on a similar base-2 factor because the images were also chosen to be resized using base-2, in the dimension of 64×64 and 128×128 so that mathematically, the dimension can be relate back to the size of the hidden layer by a similar factor. The reason for these two dimensions was obviously due to the computational complexities; when an image has a bigger dimension, it produces larger matrix and this increases computation time and also a burden for memory management.

One additional experiment using Geometric Pyramid Rule (GPR) was added at the end of each case study to see whether suggestion from literature is plausible for this neural network architecture. The number produced by GPR will not necessarily be of base-2.

For the purpose to make discussion readable in this chapter, dataset which uses 64×64 pixels dimension will be referred to as *Dataset 1* and 128×128 as *Dataset 2*. The reference figures for all four case studies can be seen in appendix A.

7.1.1 Case Study 1 – Dataset with Colour Images

The training losses for dataset 1 look smooth except for when the hidden layers were large; 1024 and 2048. 1024 had a noisy peak before it stabilised while 2048 had a huge spike before it stabilized. The losses during the training for dataset 2 has a mixture of stable and noisy. A fraction during the epoch can be seen to be noisy for 512, 768 and 1024 but otherwise stable. However 256 and 2048 created obvious noisiness.

When using GPR, it calculated 110 neurons for dataset 1 and 221 for dataset 2. Experiment using GPR in dataset 1 shows that it converge rather slowly, while for dataset 2, it seems that not only that GPR is noisy so does the hidden layer with 256 neurons.

Training accuracies for both dataset 1 and dataset 2 have high rates, except for dataset 2 when the hidden layer's neurons was 221. Similarly for testing accuracy, the results look stable, except again for dataset 2 when the hidden layer was 221.

The experiments for this case study show a good merit for application of neural network for face recognition using 3-channel colour images. In most experiments, testing accuracy stays in a range between $93 - 95\%$ and the average is 91.6%.

7.1.2 Case Study 2 – Dataset with Grayscale Images

Parameters for this case study are the same as in the previous one except the number of input features. This is due to the original colour images that have three colour channels (red, green, blue), have been reduced to grayscale images which contain only a single colour channel.

The first obvious overall observation for this case study is that how smooth gradients are for dataset 1 and dataset 2 for losses and accuracy for both training and testing. However, dataset 2 contains some element of noisiness that can be disregard as the lines are smoothing and almost overlapping on each other.

For both instances where the hidden layers' number of neurons determined using GPR were showing similar behaviour where both took a longer time to converge. This is probably because the size of the layer is quite small that the sum function produces either very large number of very small number and therefore affected by the gradient diminishing problem which was discussed in the chapter 2 with regards to Sigmoid activation function.

When the hidden layer's neurons number was changed to 2048, it was obvious for both datasets that the loss spiked before it sharply started to converge towards lower loss asymptote. This is probably because the size of the layer is too big that the summation process which used initialized weights are producing large values, similar in the case above. As the epoch goes on and backpropagation altered the weights values, then it started to converge towards the desired result.

In terms of accuracy, the trainings were showing some very good results. For dataset 2, the training shows 100% of accuracy except when hidden layer was 256 and when GPR was used. Dataset 1 shows more than 99% training accuracy except for when GPR was used. However, the mean testing accuracy for dataset 1 is only 89.94% and 91.30% for dataset 2. It is also observed that using 2048 neurons for hidden layer did not improve testing accuracy at all.

7.1.3 Case Study 3 – Dataset with LBP Images

Case study 3 shows a promising results where the 256, 512, 768, 1024 and 2048 are almost overlapping for dataset 1. Similar occurrence also for dataset 2 but noises can be seen to peak between epoch 50 and epoch 180.

Input feature's length is 3 times smaller in comparison to colour images because LBP images only have 1 colour channel. Additional information loss during the LBP calculations are for the border of each window. When 64×64 image is divided into 16×16 windows, it created 4×4-pixel dimension for each window; 4 pixels will have their LBP calculated, and the other 12 neighbours will turn to 0. In total, there will be $12 \times 256 = 3,072$ pixels with zero values, which represent a loss of 75%.

When 128×128 image is divided into 16×16 windows, it created 8×8-pixel dimension for each window; 36 pixels will have their LBP calculated, and the other 28 neighbours will turn to 0. In total, there will be $28 \times 256 = 7,168$ pixels with zero values, which is a loss of 43.75%. These *losses* however can still be seen on LBP image as black as the area has no colour intensity.

When the hidden layer neurons were calculated using GPR, the values are lower than 256. It can be seen that both losses for dataset 1 and dataset 2 are converging very slowly but later overlapping with the other losses on the graph. Hidden layer with 2048 neurons for both datasets peak in the early training epoch before sharply converging.

All twelve experiments have the loss values converging quite close to zero, that show the predictions are getting better for subsequent epoch. Also can be seen that the training accuracy for all 12 experiments are 100%. But when compared against the testing accuracy, there seems to be an almost equal *loss* of accuracy rate as the gap is around 20% for all twelve with the mean 81.47% for dataset 1 is and 80.75% for dataset 2. The losses can be attributed to the loss of information when LBP process took place, but despite the differences in the magnitude of losses for both datasets, the accuracies for testing are converging towards 80. When plotted, both training and testing accuracy are showing smooth lines which are converging towards similar horizontal asymptotes, $L(Y, \hat{Y}) \rightarrow 0$ for loss and $\gamma_{testing}$ has its accuracy $\rightarrow 80$.

Training accuracy always reaches 100% which suggests that overfitting takes place in the neural network. Testing accuracy however only reaches $79.5 - 82.1\%$, which further strengthens the argument of overfitting. Testing accuracy does not correlate with increases in layer size nor with image dimension. Although, usually testing accuracy tends to be a bit better with smaller image dimension.

7.1.4 Case Study 4 – Dataset with LBP Histograms

This case study is the most interesting in terms of observations as the expectation was somewhat different. It was hypothesized that if important information is extracted and organized, it should provide a better input for neural network because it has an organized pattern to learn. Therefore, it was expected that a neural network to perform rather quickly and with a better accuracy.

But the observations from 12 experiments show that the losses are not efficient enough to produce a concrete training accuracy. For dataset 1, the loss flat-lined with almost zero gradient which showed that there was no more possibility to correct the losses produced by the output layer. Similar case can also be observed for dataset 2, however the line is showing a small gradient which shows a behaviour that after a certain longer epoch, it might converge closer to zero. As to prove this is the case, an additional experiment was done using similar variables and input from dataset 2, but the epoch was changed to $10,000$ that is 5 times more than with the previous experiments.

L1-Size	**Image Dimension**	**Training Time**	**Loss**	**Accuracy (Training) %**	**Accuracy (Testing) %**
1,024	128×128	6:22:15	0.5088	94.06	87.36

Table 12 : Result for supplement experiment 13 for case study 4.

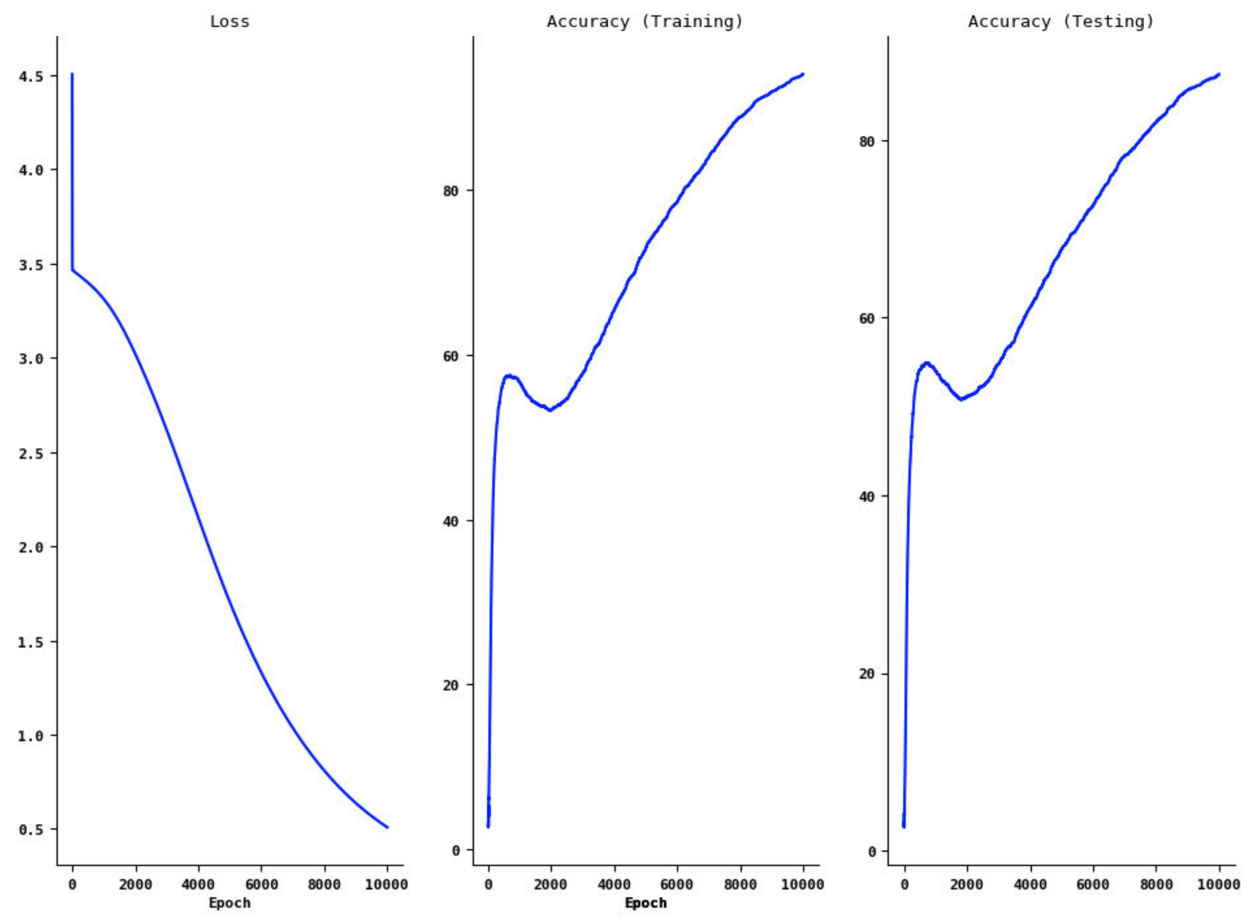


Figure 44 :Experiment 13 result shows that if training epochs are increased, the loss sinks gradually. Similarly, for accuracy rate, it rises quite steep before dropping and rising again gradually.

It can be observed that LBPH takes a longer epoch and there is a sink after a sharp spike at around epoch $1,500$ and spiking again at around epoch $2,200$. The accuracy for both training and testing shows a possibility for the trained model to be used for identification.

The accuracy for both training and testing seem to be lingering between 40 and 50%. As these values are the barometer of the quality of trained model, it can be assumed that it is not suitable for further use of identification.

The training time for dataset 1 and dataset 2 for each experiment with different hidden layer neurons are almost similar because despite the difference in terms of image dimension in both datasets, when converted to histogram both are having similar quantity of concatenated histograms.

7.2 CASE STUDY ANALYSIS

Comparison will be made by comparing case study 1 and case 2 because they have similarities except for the number of colour channels; case study 2 and case study 3 due to the fact they are both single channel; as well as case 3 and case study 4 because they are both went through the LBP stage. Then the analysis for all case studies will follow in terms of time, accuracy and loss function.

The first two analysis will focus on cross entropy loss. This is the measurement to see the distance between the predicted output and target output is calculated using Softmax Cross Entropy Loss as explored in chapter 2. The closer the value is to zero, then the closer of each predicted output to each target output.

7.2.1 Case Study 1 and Case Study 2

Case study 1 and case study 2 differ in terms of the number of colour channels for images in the dataset that are then being used as the input features. The latter has only a third of input features when compared to the former. Despite the reduction in size, it did not reflect badly in term of accuracy where the testing accuracy for case study 2 has a very slight difference.

Dataset	**Case Study**	**256**	**512**	**768**	**1024**	**2048**
1	1	0.0503	0.0330	0.0268	0.0266	0.0246
	2	0.1164	0.1062	0.0963	0.0929	0.0845
2	1	0.0451	0.0223	0.0165	0.0165	0.0153
	2	0.0492	0.0349	0.0322	0.0300	0.0303

Table 13 : Cross entropy loss for both dataset for each case study at epoch 2,000.

From the table above, it is quite clear that dataset 2 performs better in terms of cross entropy loss and the values are in the range of 10^{-2}.

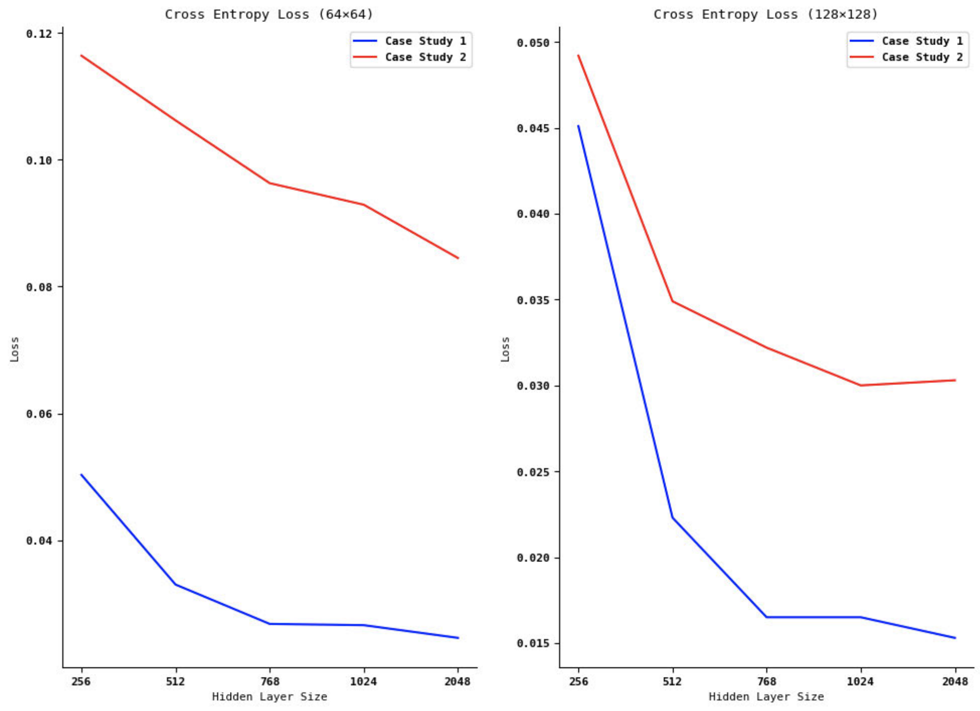


Figure 45 : Cross entropy loss for both case study 1 and case study 2. The graph of the right is for dataset 1 and on the left for dataset 2.

Case Study	**256**	**512**	**768**	**1024**	**2048**
1	90	68	62	62	62
2	42	33	33	32	36

Table 14 : Performance in terms of percentage proportion between dataset, calculated as $\frac{Dataset\ 2}{Dataset\ 1} \times 100$

When comparing in terms of percentage proportion, to find one with the smallest value for the magnitude of loss reduction, the best performance is shown by case study 2 when dataset 2 with 768 neurons in the hidden layer. The loss value is 33% which means a loss reduction of 67%. Despite 1024 neurons in the hidden layer for dataset 2 in case study 2 has the smallest number, but in term of time, it took an hour longer which is also a factor that must considered.

7.2.2 Case Study 2 and Case Study 3

These two case studies share one similarity, where the former use one-colour channel and the latter, resulted from LBP process that keeps the original one-colour channel. Therefore, both have the same number of input features.

Dataset	**Case Study**	**64**	**128**	**256**	**512**	**768**	**1024**	**2048**
1	2	0.1939	None	0.1164	0.1062	0.0963	0.0929	0.0845
	3	0.0589	None	0.0305	0.0258	0.0239	0.0226	0.0209

2	2	None	0.0875	0.0492	0.0349	0.0322	0.0300	0.0303
	3	None	0.0257	0.0151	0.0112	0.0099	0.0092	0.0094

Table 15 : Cross entropy loss for both dataset for each case study at epoch 2,000.

From the table above, it is quite clear that dataset 2 performs better in terms of cross entropy loss and the values are in the range of 10^{-2} and case study 3 in the range of 10^{-3}.

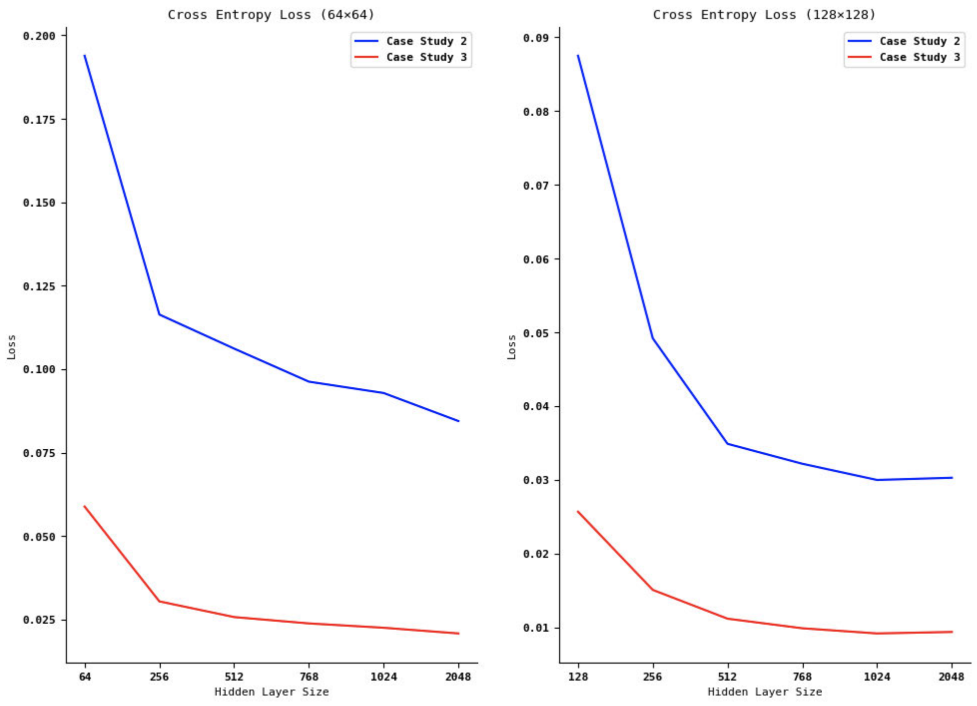


Figure 46 : Cross entropy loss for both case study 2 and case study 3. The graph of the right is for dataset 1 and on the left for dataset 2.

The general observation from the figure above for approximation of the predicted output against the target output, case study 3 has a superior cross entropy loss value. It means that the training for case study 3 is much faster and much closer to the target output.

Case Study	256	512	768	1024	2048
2	42	33	33	32	36
3	50	43	41	41	45

Table 16 : Performance in terms of percentage proportion between dataset, calculated as $\frac{Dataset\ 2}{Dataset\ 1} \times 100$

When comparing in terms of percentage proportion, to find one with the smallest value for the magnitude of loss reduction, the best performance is shown by case study 2 when dataset 2 with 768 neurons in the hidden layer. The loss value is only 33% which means a reduction of 67%. Despite case study 3 performs better in terms of time complexity, the speed is not by much and considerations were also put on the testing accuracy which clearly shows that case study 2 is superior.

7.2.3 Case Study 3 and Case Study 4

Case study 3 and case study 4, despite using almost similar step for pre-processing to produce input features for neural network but the results are showing difference performance. Case study 3 has smaller input features compared to case study 4, but it has better accuracy in terms of training and testing. Case study 4 probably lost too much information because unlike LBP images which only involves two steps; first transforming to grayscale and then LBP is calculated for each pixel; LBP histograms involves additional two additional steps; where step three is frequency calculations for local histogram and followed by step four that is concatenating all local histograms to form one final histogram. This may also explain the loss important information for input features. It also must be noted that the input features in the case study 4 have not been normalized and therefore is not suitable with Sigmoid activation function.

Case study 4 was showing different results when compared to the previous three. This is probably because the neural network was using Sigmoid as its activation function for the hidden layer. Sigmoid works best for values in $[-5, 5]$ and will start showing the sign of gradient diminishing when values are getting bigger or too small. In the first three case studies, values were standardized and scaled to be within $[0, 1]$ while for the last case study, the value has not been standardized and in the range of $[0,\ max(6464)]$ or $[0,\ max(128128)]$. The last case study might benefit from ReLU as its activation function for the hidden layer.

7.2.4 Training Time

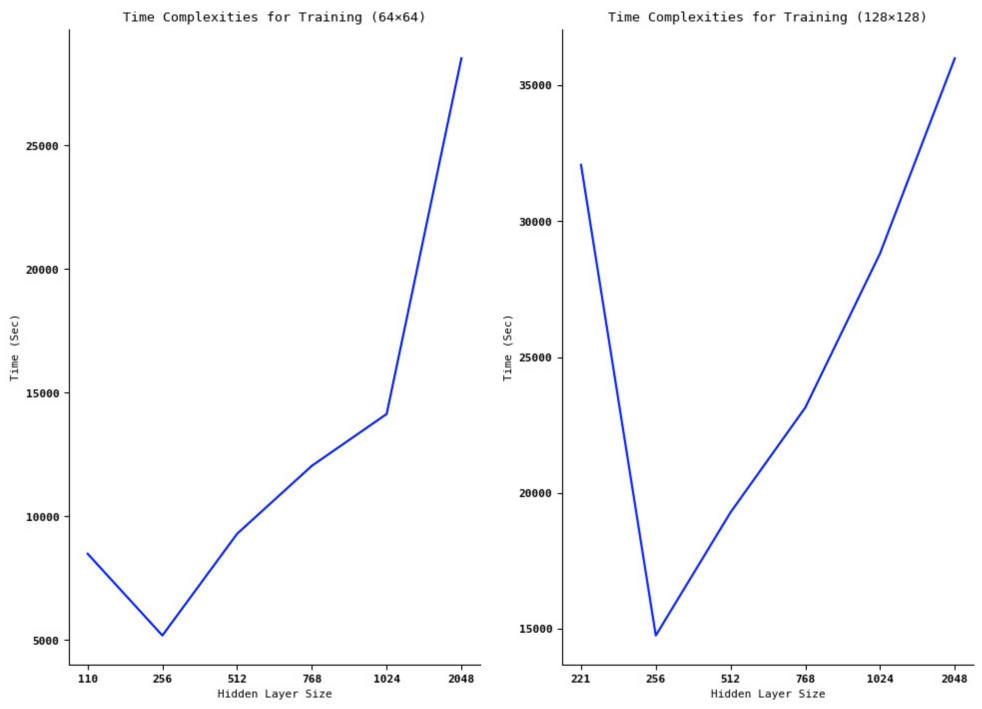


Figure 47 : Training time for case study 1. The graph on the left is for dataset 1 and graph on the right is for dataset 2.

The first obvious peculiarity seen in both figures are the time complexity for both hidden layers which had the number of their neurons determined using GPR. Despite both being smaller than the subsequent step of hidden layer, they both took longer training time. Other than that, it can be observed for hidden layers with neurons $256, 512, 1024$ and 2048 are on the trending growth. For dataset 1, the first three has a steady slope before sharply rising. For dataset 2, the time complexity looks almost linear.

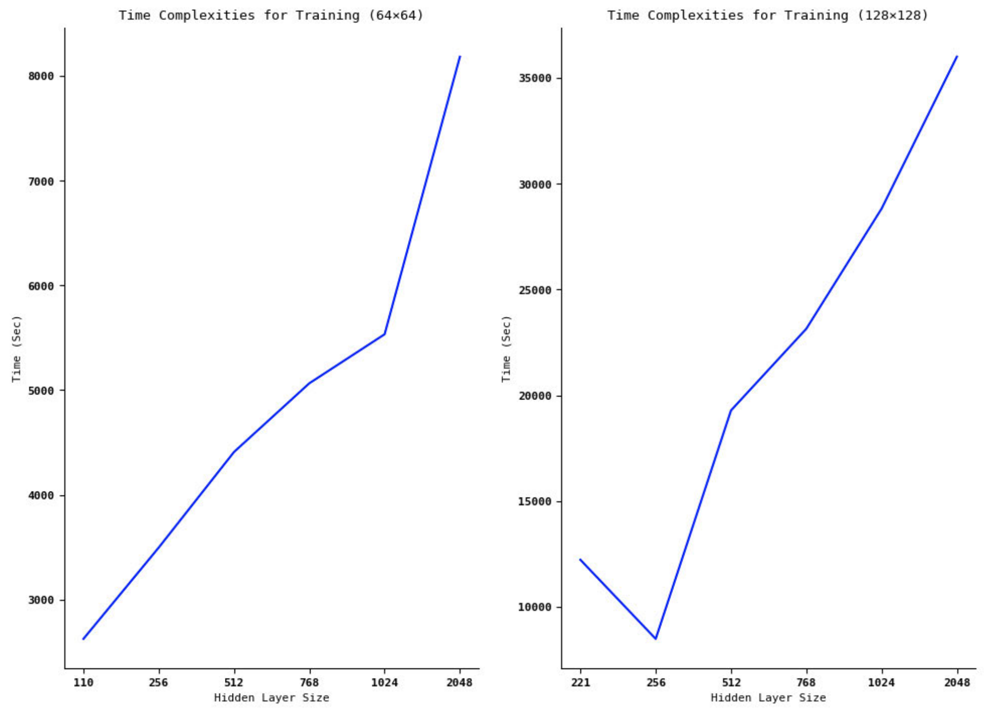


Figure 48 : Training time for case study 2. The graph on the left is for dataset 1 and graph on the right is for dataset 2.

In this case study, the time complexity when hidden layer neurons were determined using GPR seems only to affect for dataset 2. For dataset 1, the first 5 experiments are showing an almost linear time growth but after 1024, it started to rise sharply. However, for dataset 2, it is exhibiting similar traits with case study 1, from 256 onwards, it has an almost linear time growth.

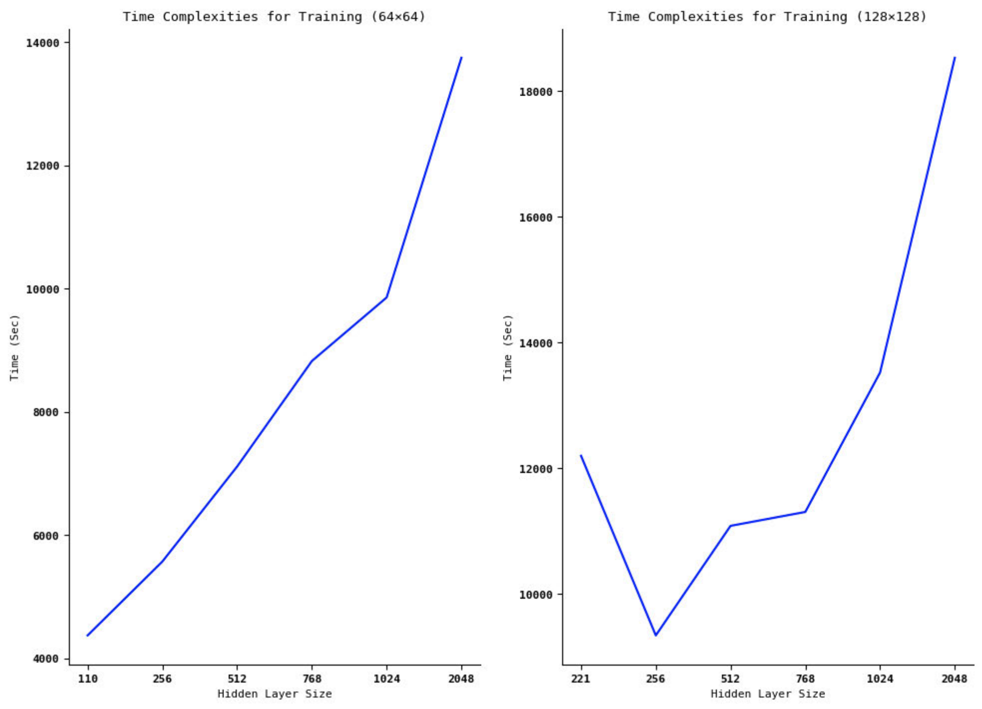


Figure 49 : Training time for case study 3. The graph on the left is for dataset 1 and graph on the right is for dataset 2.

Case study 3 is exhibiting similar traits to case study 2 where dataset 1 is showing time growth as almost linear except that the gradient of the slope is quite high and from 1024 to 2048, the gradient is even higher. For dataset 2, a similar trend can be observed with case study 2 where GPR is affecting the time complexity and the growth, if not because of the almost low gradient between 512 and 768, it can be assumed to be an almost linear. The traits between case study 2 and case study 3 are probably due to their similar nature; case study 2 has its dataset in grayscale, and case study 3 has its dataset in LBP where both are a single channel with pixel that has a value between $[0, 255]$.

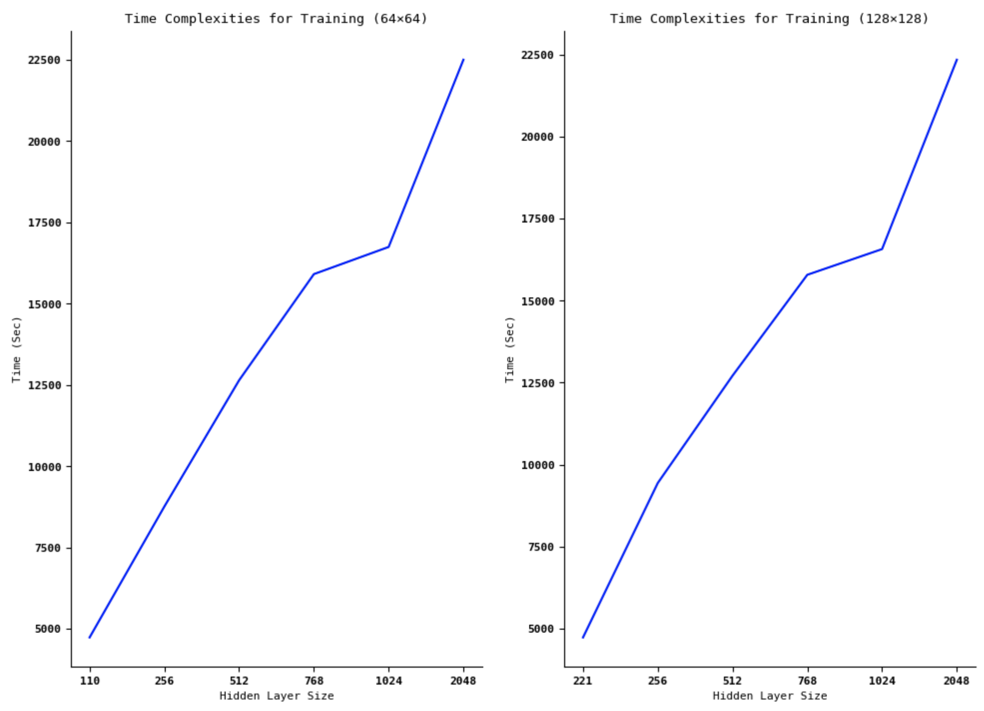


Figure 50 : Training time for case study 4. The graph on the left is for dataset 1 and graph on the right is for dataset 2.

Interesting to observe for case study 4 that the graphs are almost plotting on similar point. This is probably because of both datasets are having the same number of input features. If the low gradient between 768 and 1024 are ignored, then both graphs might show an almost linear time growth.

From all four figures, it can be seen that time growth are showing a trait of a linear function. It can also be seen in the first three case studies that training using hidden layer with its neurons determined using GPR is not to be relied upon.

7.2.5 Testing and Training Accuracy

Averaged training and testing accuracy for all four case studies are summarized in the table below.

Case Study	Dataset 1		Dataset 2	
	Training %	Testing %	Training %	Testing %
1	99.99	93.59	96.08	89.60
2	99.34	89.94	99.99	91.30
3	100	81.47	100	80.75
4	56.32	49.61	51.85	49.61

Table 17 : Summary of averaged training and testing for both dataset 1 and dataset 2.

From the table above, it can be seen that for dataset 1, case study 1 performs better with almost 100% training accuracy and highest testing accuracy among all case studies. When looking at the table, the testing accuracy are going down from one case study to another. This is probably because of not only that the image in the dataset is small and when transformed to grayscale, LBP image and LBP histograms; the already small number of information are getting lost along the way.

Considerations must also be thrown in as the image dimension for dataset 1 is rather small and a lot of details are missing and for the purpose of face recognition, this may not be a good thing. Therefore, by considering dataset 2 which has twice the size of image dimension, better image quality and along with it, the details may be preserved and better for recognizing faces. When considering dataset 2, case study 2 seems to show the best performance where the training accuracy is almost 100% and has the highest testing accuracy.

7.2.6 Hidden Layer Size and Gradient Descent

By going through losses figures from case studies 1, 2 and 3, it is consistently seen that a neural network with a smaller size of the hidden layer requires more epochs and therefore more training time to achieve the same value of loss function or accuracy. Gradient descent, as was explained in chapter 2, makes up for the phenomenon of convergence in this implementation of neural network. Thus, it can be said that gradient descent seeks the global minimum faster in neural networks with a higher number of neurons in the hidden layer. Or, in other words, the method of gradient descent becomes more efficient and yields good results more quickly if the hidden layer is equipped with a sufficient number of neurons. However, it should be noted that in every case study, a hidden layer with 2048 neurons always resulted in a huge spike in values of the loss function (that equals to strong divergence) prior to fast convergence. As such a spike of initial divergence is undesirable and could be confusing under some circumstances, tt can be concluded that in terms of fast convergence, best results were yielded with hidden layers that contained 1024 neurons. More or less than that resulted in less efficient gradient descent.

7.3 CONCLUSION

This project has achieved to answer its main research question by providing theory and analysis for *neural network* in chapter 2, *face recognition* in chapter 3 as well as *dataset* in chapter 4. The implementation in chapter 5 were using the knowledge acquired in the three previous chapters.

This report explored and explained about forward propagation, backpropagation and optimization for basic functioning of an artificial network. A representative selection of simple mathematical formulas was shed more light on, like activation and loss functions. All the underlying mathematical concepts necessary for neural networks that were explored in this report were later implemented for face recognition while OOP, UML and MVC models were followed. The code was written in Python and twelve experiments for each case study with four different case studies were performed to analyse the behaviour and performance of implemented Artificial Neural Network (ANN). Testing accuracy of face recognition was in most of the experiments are above 90% and in few cases reached nearly 95%. This said, semester binding has been clearly satisfied. However, the plan to implement Convolutional Neural Network (CNN) did not materialize due to the time constraint.

A quick observations from all 49 experiments, it can be concluded that ANN has the capability for face recognition; and based on the analysis from all four case studies, it can be concluded that the grayscale dataset is showing the most smooth for loss and accuracy graphs and dataset with image dimension 128×128 with a total of 768 for the size hidden layer neurons is has the best result in terms of accuracy, time complexity and performance.

It must also be noted that the Geometric Pyramid Rule which during analysis seems promising, but when implemented, shows catastrophic result that not only it has the lowest score for accuracy by also increased the time complexity.

To bind the conclusion up; this report has achieved its aims and its target to explore the mathematical concepts behind neural network and using the knowledge to code an Artificial Neural Network and implement for face recognition. By conducting a number of experiments using the implemented ANN, a concrete and conclusive evidence on choosing the image dimension and the size of neurons in the hidden layer for optimum has been determined using numerical comparison.

7.4 FUTURE POSSIBLE RESEARCH DIRECTIONS

This research has a lot of room for improvement. For example, due to time constraints and despite being tested during implementation, other activation functions for example hyperbolic tangent and ReLU were not used during experimentation. Besides, LBPH may benefit in terms of better result if ReLU was implemented instead of Sigmoid.

As the time was quite short to gather enough facial images, finding volunteers and building the dataset, it was tested against a relatively small number of volunteers. It can be proposed to study the time complexity when the dataset has 50 volunteers and have it compared for a dataset with 100 volunteers to see their correlation as well as accuracy score.

Another aspect that may be considered is to use a different optimization method, for example Mini-Batch Gradient, considering that every volunteer has 200 images. As the number of volunteers grow, so will the images in the dataset. It will be quite inefficient to load all of the images into the memory and this will create memory complexity. By using Mini-Batch Gradient, the optimization will be able to speed up calculations in the memory, on the argument that each batch will have a limit of maximum number of images for calculations.

In terms of reducing the time complexity, the implementation can be reworked using Graphics Processing Unit (GPU) instead of standard CPU for training. This implementation may also be conducive in implementing Convolutional Neural Network (CNN). A lot of literature is pointing towards CNN for image classification as it performs better than ANN. The way forward is to explore CNN and followed by different case studies but utilizing the same dataset, so that the performance between ANN and CNN can be compared on a fair ground.

R E F E R E N C E S

[1] B. Müller, J. Reinhardt, and M. T. Strickland, *Neural Networks: An Introduction*. Springer Science & Business Media, 1995.
[2] "The Nature of Code." [Online]. Available: https://natureofcode.com/book/chapter-10-neural-networks/. [Accessed: 16-Apr-2019]
[3] R. Brandom, "Amazon is selling police departments a real-time facial recognition system," *The Verge*, 22-May-2018. [Online]. Available: https://www.theverge.com/2018/5/22/17379968/amazon-rekognition-facial-recognition-surveillance-aclu. [Accessed: 11-Feb-2019]
[4] D. Wilkins, "Enter the UAE with just your face," *Time Out Dubai*, 11-Oct-2018. [Online]. Available: https://www.timeoutdubai.com/dubai-airport/386146-enter-the-uae-with-just-your-face. [Accessed: 19-Feb-2019]
[5] Editorial Team, "CaixaBank rolls out facial recognition at the ATM," *Finextra Research*, 14-Feb-2019. [Online]. Available: https://www.finextra.com/newsarticle/33388/caixabank-rolls-out-facial-recognition-at-the-atm. [Accessed: 16-Feb-2019]
[6] B. Poulsen, "Supervisor Presentation." 02-2019.
[7] Techopedia Staff, "INFOGRAPHIC: Artificial Intelligence vs Machine Learning vs Deep Learning," *Techopedia.com*. [Online]. Available: https://www.techopedia.com/infographic-artificial-intelligence-vs-machine-learning-vs-deep-learning/2/33882. [Accessed: 19-Apr-2019]
[8] M. Yao, "12 Amazing Deep Learning Breakthroughs of 2017," *Forbes*, 05-Feb-2018. [Online]. Available: https://www.forbes.com/sites/mariyayao/2018/02/05/12-amazing-deep-learning-breakthroughs-of-2017/. [Accessed: 07-Mar-2019]
[9] K. Gates, *Our Biometric Future: Facial Recognition Technology and the Culture of Surveillance*. NYU Press, 2011.
[10] "Face Recognition Based on Euclidean Distance and Texture Features - IEEE Conference Publication." [Online]. Available: https://ieeexplore.ieee.org/document/6642978. [Accessed: 20-Feb-2019]
[11] Ars Staff, "The basics of modern AI—how does it work and will it destroy society this year?," *Ars Technica*, 09-Apr-2019. [Online]. Available: https://arstechnica.com/features/2019/04/from-ml-to-gan-to-hal-a-peak-behind-the-modern-artificial-intelligence-curtain/. [Accessed: 01-May-2019]
[12] S. Shankland, "Boosted by AI, facial recognition eases our path through an increasingly digital world," *CNET*, 28-Mar-2019. [Online]. Available: https://www.cnet.com/news/huge-leaps-in-ai-have-made-facial-recognition-smarter-than-your-brain/. [Accessed: 28-Mar-2019]
[13] T. M. Mitchell, *Machine Learning*. McGraw-Hill Education, 1997.
[14] L. Bermudez, "Overview of Artificial Intelligence Buzz," *Medium*, 22-Oct-2018. [Online]. Available: https://medium.com/latinxinai/overview-of-artificial-intelligence-buzz-71280b588f. [Accessed: 19-Apr-2019]
[15] A. Saiyad, "Let's Dive in the World of Machine Learning," *Medium*, 11-Apr-2019. [Online]. Available: https://medium.com/yudiz-solutions/lets-dive-in-the-world-of-machine-learning-eef6be9eb0d6. [Accessed: 01-May-2019]
[16] S. W. Mindy Weisberger, "AI Is Good (Perhaps Too Good) at Predicting Who Will Die Prematurely," *Live Science*, 27-Mar-2019. [Online]. Available: https://www.livescience.com/65087-ai-premature-death-prediction.html. [Accessed: 30-Mar-2019]
[17] M. Preziuso, "1. Machine Learning 101 - An introduction," *Towards Data Science*, 30-Dec-2018. [Online]. Available: https://towardsdatascience.com/machine-learning-101-part-1-an-introduction-494cb654d50b. [Accessed: 19-Apr-2019]

[18] A. Phadnis, “Machine Learning — Teaching Machines to Learn,” *Good Audience*, 08-Oct-2018. [Online]. Available: https://blog.goodaudience.com/machine-learning-teaching-machines-to-learn-61b0f3eaaba8. [Accessed: 09-Apr-2019]
[19] J. P. Mueller and L. Massaron, *Machine Learning For Dummies*. John Wiley & Sons, 2016.
[20] D. Rothman, *Artificial Intelligence by Example: Develop Machine Intelligence from Scratch Using Real Artificial Intelligence Use Cases*. Packt Publishing, 2018.
[21] C. Zoltan, “The Truth about Neural Networks,” *Medium*, 13-Nov-2018. [Online]. Available: https://medium.com/stupid-simple-ai-series/the-truth-about-neural-networks-70ec055d1e9b. [Accessed: 08-Apr-2019]
[22] “The Nature of Code.” [Online]. Available: https://natureofcode.com/book/chapter-10-neural-networks/. [Accessed: 16-Apr-2019]
[23] J. Brownlee, “How to Improve Performance With Transfer Learning for Deep Learning Neural Networks,” *Machine Learning Mastery*, 07-Feb-2019. [Online]. Available: https://machinelearningmastery.com/how-to-improve-performance-with-transfer-learning-for-deep-learning-neural-networks/. [Accessed: 14-Apr-2019]
[24] Elvis, “A Light Introduction to Transfer Learning for NLP,” *Medium*, 26-Jul-2018. [Online]. Available: https://medium.com/dair-ai/a-light-introduction-to-transfer-learning-for-nlp-3e2cb56b48c8. [Accessed: 19-Apr-2019]
[25] “A Beginner’s Guide to Linear Regression in Python with Scikit-Learn.” [Online]. Available: https://www.kdnuggets.com/2019/03/beginners-guide-linear-regression-python-scikit-learn.html. [Accessed: 30-Mar-2019]
[26] S. Remanan, “Linear Regression using Python,” *Towards Data Science*, 04-Sep-2018. [Online]. Available: https://towardsdatascience.com/linear-regression-using-python-ce21aa90ade6. [Accessed: 30-Mar-2019]
[27] garvitanand, “TOP 10 Machine Learning Algorithms,” *Good Audience*, 10-Feb-2019. [Online]. Available: https://blog.goodaudience.com/top-10-machine-learning-algorithms-2a9a3e1bdaff. [Accessed: 14-Apr-2019]
[28] “Looking inside neural nets.” [Online]. Available: https://ml4a.github.io/ml4a/looking_inside_neural_nets/. [Accessed: 02-Apr-2019]
[29] “comp.ai.neural-nets FAQ, Part 3 of 7: GeneralizationSection - How many hidden layers should I use?” [Online]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-9.html. [Accessed: 30-Mar-2019]
[30] J. Heaton, *Introduction to Neural Networks with Java*. Heaton Research, Inc., 2008.
[31] “Training an Artificial Neural Network - Intro,” *solver*, 23-Mar-2012. [Online]. Available: https://www.solver.com/training-artificial-neural-network-intro. [Accessed: 30-Mar-2019]
[32] P. Chavan, “How to decide the number of hidden layers and nodes in a hidden layer?,” *ResearchGate*, 03-Sep-1995. [Online]. Available: https://www.researchgate.net/post/How_to_decide_the_number_of_hidden_layers_and_nodes_in_a_hidden_layer. [Accessed: 30-Mar-2019]
[33] T. Masters, *Practical Neural Network Recipes in C++*. Morgan Kaufmann, 1993.
[34] “Looking inside neural nets.” [Online]. Available: https://ml4a.github.io/ml4a/looking_inside_neural_nets/. [Accessed: 02-Apr-2019]
[35] “To Study Implementation of Gradient Descent for Multi-class Classification Using a SoftMax Regression and Neural Networks.” [Online]. Available: https://rstudio-pubs-static.s3.amazonaws.com/337306_79a7966fad184532ab3ad66b322fe96e.html. [Accessed: 05-Apr-2019]
[36] Z. Hao, “Weight Initialization Methods in Neural Networks | Isaac Changhau,” *Isaac Changhau*, 24-May-2017. [Online]. Available:

https://isaacchanghau.github.io/post/weight_initialization/. [Accessed: 16-Mar-2019]

[37] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer (India) Private Limited, 2013.

[38] N. Chandarana, “Classification: A Linear Approach (Part 1),” *Towards Data Science*, 18-Apr-2019. [Online]. Available: https://towardsdatascience.com/classification-a-linear-approach-part-1-b080c13992dd. [Accessed: 01-May-2019]

[39] “A ‘Brief’ History of Neural Nets and Deep Learning,” *Andrey Kurenkov’s Web World*. [Online]. Available: http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/. [Accessed: 07-Apr-2019]

[40] A. S. Walia, “Activation functions and it’s types-Which is better?,” *Towards Data Science*, 29-May-2017. [Online]. Available: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f. [Accessed: 08-Apr-2019]

[41] V. Nigam, “Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning,” *Towards Data Science*, 11-Sep-2018. [Online]. Available: https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e90e0a90. [Accessed: 01-May-2019]

[42] S. Siddiqui, “How would we find a better activation function than ReLU?,” *Medium*, 01-Jan-2019. [Online]. Available: https://medium.com/shallow-thoughts-about-deep-learning/how-would-we-find-a-better-activation-function-than-relu-4409df217a5c. [Accessed: 08-Apr-2019]

[43] “Deep Learning.” [Online]. Available: https://www.deeplearningbook.org/. [Accessed: 24-May-2019]

[44] K. Y. Lee, “[Personal Notes] Deep Learning by Andrew Ng — Course 1: Neural Networks and Deep Learning,” *Medium*, 03-Mar-2019. [Online]. Available: https://medium.com/@keonyonglee/bread-and-butter-from-deep-learning-by-andrew-ng-course-1-neural-networks-and-deep-learning-41563b8fc5d8. [Accessed: 01-May-2019]

[45] N. Shibuya, “Demystifying Cross-Entropy,” *Towards Data Science*, 28-Oct-2018. [Online]. Available: https://towardsdatascience.com/demystifying-cross-entropy-e80e3ad54a8. [Accessed: 01-May-2019]

[46] D. Godoy, “Understanding binary cross-entropy / log loss: a visual explanation,” *Towards Data Science*, 21-Nov-2018. [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a. [Accessed: 01-May-2019]

[47] “To Study Implementation of Gradient Descent for Multi-class Classification Using a SoftMax Regression and Neural Networks.” [Online]. Available: https://rstudio-pubs-static.s3.amazonaws.com/337306_79a7966fad184532ab3ad66b322fe96e.html. [Accessed: 05-Apr-2019]

[48] “Single-Layer Neural Networks and Gradient Descent,” *Dr. Sebastian Raschka*, 24-Mar-2015. [Online]. Available: https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html. [Accessed: 31-Mar-2019]

[49] “5 Training Hidden Units with Back Propagation,” 16-Dec-2015. [Online]. Available: https://web.stanford.edu/group/pdplab/pdphandbook/handbookch6.html. [Accessed: 31-Mar-2019]

[50] “5 Training Hidden Units with Back Propagation,” 16-Dec-2015. [Online]. Available: https://web.stanford.edu/group/pdplab/pdphandbook/handbookch6.html. [Accessed: 31-Mar-2019]

[51] “Deep Learning.” [Online]. Available: https://www.deeplearningbook.org/. [Accessed: 24-May-2019]

[52] "[No title]." [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/asr/2015-16/asrX1-nn.pdf. [Accessed: 31-Mar-2019]
[53] "Overfitting in Machine Learning: What It Is and How to Prevent It," *EliteDataScience*, 07-Sep-2017. [Online]. Available: https://elitedatascience.com/overfitting-in-machine-learning. [Accessed: 14-Apr-2019]
[54] A. Bronshtein, "Train/Test Split and Cross Validation in Python," *Towards Data Science*, 17-May-2017. [Online]. Available: https://towardsdatascience.com/train-test-split-and-cross-validation-in-python-80b61beca4b6. [Accessed: 01-May-2019]
[55] A. Gebhart, "Facial recognition: Apple, Amazon, Google and the race for your face," *CNET*, 18-Mar-2019. [Online]. Available: https://www.cnet.com/news/facial-recognition-apple-amazon-google-and-the-race-for-your-face/. [Accessed: 18-Mar-2019]
[56] "Face Detection vs. Facial Recognition," *www.SecurityInfoWatch.com*. [Online]. Available: https://www.securityinfowatch.com/access-identity/biometrics/article/10627859/the-difference-between-face-detection-analytics-and-facial-recognition-biometrics. [Accessed: 12-Apr-2019]
[57] L. Wang and D.-C. He, "Texture classification using texture spectrum," *Pattern Recognit.*, vol. 23, no. 8, pp. 905–910, Aug. 1990.
[58] T. Ojala, M. Pietikainen, and D. Harwood, "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions," in *Pattern Recognition, 1994. Vol. 1 - Conference A: Computer Vision & Image Processing., Proceedings of the 12th IAPR International Conference on*, 1994, vol. 1, pp. 582–585 vol.1.
[59] "[No title]." [Online]. Available: https://arxiv.org/pdf/1611.09099v1.pdf. [Accessed: 11-May-2019]
[60] "An HOG-LBP human detector with partial occlusion handling - IEEE Conference Publication." [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5459207. [Accessed: 13-Apr-2019]
[61] K. S. do Prado, "Face Recognition: Understanding LBPH Algorithm," *Towards Data Science*, 10-Nov-2017. [Online]. Available: https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b. [Accessed: 11-May-2019]
[62] "Face Detection vs. Facial Recognition," *www.SecurityInfoWatch.com*. [Online]. Available: https://www.securityinfowatch.com/access-identity/biometrics/article/10627859/the-difference-between-face-detection-analytics-and-facial-recognition-biometrics. [Accessed: 12-Apr-2019]
[63] T. H. Le, "Applying Artificial Neural Networks for Face Recognition," *Advances in Artificial Neural Systems*, vol. 2011, Nov. 2011 [Online]. Available: https://www.hindawi.com/journals/aans/2011/673016/abs/. [Accessed: 23-Mar-2019]
[64] P. J. Grother, M. L. Ngan, and K. K. Hanaoka, "Ongoing Face Recognition Vendor Test (FRVT) Part 2: Identification," NIST Interagency/Internal Report (NISTIR) - 8238, Nov. 2018 [Online]. Available: https://doi.org/10.6028/NIST.IR.8238. [Accessed: 28-Mar-2019]
[65] A. Kortylewski, A. Schneider, T. Gerig, B. Egger, A. Morel-Forster, and T. Vetter, "Training Deep Face Recognition Systems with Synthetic Data," 16-Feb-2018 [Online]. Available: http://arxiv.org/abs/1802.05891. [Accessed: 01-Apr-2019]
[66] S. Ghoneim, "5 Steps to correctly prepare your data for your machine learning model," *Towards Data Science*, 07-Apr-2019. [Online]. Available: https://towardsdatascience.com/5-steps-to-correctly-prep-your-data-for-your-machine-learning-model-c06c24762b73. [Accessed: 08-Apr-2019]
[67] J. Brownlee, "How to Evaluate Pixel Scaling Methods for Image Classification With

Convolutional Neural Networks," *Machine Learning Mastery*, 26-Mar-2019. [Online]. Available: https://machinelearningmastery.com/how-to-evaluate-pixel-scaling-methods-for-image-classification/. [Accessed: 27-Mar-2019]
[68] "IBM Knowledge Center." [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS88XH_1.6.0/iva/int_i2frs_intro.html. [Accessed: 23-Mar-2019]
[69] R. Browne, "IBM hopes 1 million faces will help fight bias in facial recognition," *CNBC*, 29-Jan-2019. [Online]. Available: https://www.cnbc.com/2019/01/29/ibm-releases-diverse-dataset-to-fight-facial-recognition-bias.html. [Accessed: 23-Mar-2019]
[70] BBC News, "IBM used Flickr photos for AI training," *BBC News*, 13-Mar-2019. [Online]. Available: https://www.bbc.com/news/technology-47555216. [Accessed: 23-Mar-2019]
[71] P. Griffin, "Understanding The Face Image Format Standards," Apr-2005. [Online]. Available: https://www.nist.gov/sites/default/files/documents/2016/12/12/griffin-face-std-m1.pdf. [Accessed: 13-Apr-2019]
[72] M. Savvides, B. V. Vijaya, and P. Khosla, "Face verification using correlation filters," Jan. 2002 [Online]. Available: http://dx.doi.org/. [Accessed: 28-Mar-2019]
[73] J. Brownlee, "How to Configure Image Data Augmentation When Training Deep Learning Neural Networks," *Machine Learning Mastery*, 11-Apr-2019. [Online]. Available: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/. [Accessed: 13-Apr-2019]
[74] J. Vincent, "Facebook is using AI to map population density around the world," *The Verge*, 09-Apr-2019. [Online]. Available: https://www.theverge.com/2019/4/9/18301738/facebook-artificial-intelligence-ai-map-world-population-density-humanitarian-efforts. [Accessed: 13-Apr-2019]
[75] "Facial Recognition Is Accurate, if You're a White Guy," 09-Feb-2018. [Online]. Available: https://www.nytimes.com/2018/02/09/technology/facial-recognition-race-artificial-intelligence.html. [Accessed: 17-Feb-2019]
[76] I. Sommerville, *Software Engineering, Global Edition*. Pearson Higher Ed, 2016.
[77] "A terrific Model View Controller (MVC) diagram | alvinalexander.com." [Online]. Available: https://alvinalexander.com/uml/uml-model-view-controller-mvc-diagram. [Accessed: 19-May-2019]
[78] K. Fakhroutdinov, "UML 2.5 Diagrams Overview," 01-Jan-2013. [Online]. Available: https://www.uml-diagrams.org/uml-25-diagrams.html. [Accessed: 19-May-2019]
[79] "[No title]." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.6924&rep=rep1&type=pdf. [Accessed: 23-May-2019]

APPENDIX

Ⓐ

Figures from Case Studies' Experiments

1.0 SUPPLEMENTAL EXPERIMENT

This is a supplemental experiment for case study 1, where all the variables are fixed but the number of epochs was halved. Instead of 2,000 this experiment will consider only using 1,000 epochs. The purpose of this experiment is to compare the training time, loss and accuracy for both training and testing.

1.1 Fixed Variables

Neural network architecture	3-Layer
Learning rate, α	0.075
Total dataset for training	3,200
Total classification	32
Number of input features (64×64×3)	12,288
Number of input features (128×128×3)	49,152

Table 18 : Variables used in this experiment.

1.2 Results

L1-Size	**Image Dimension**	**Training Time**	**Loss**	**Accuracy (Training) %**	**Accuracy (Testing) %**
256	64	0:41:11	0.1042	99.78	92.75
	128	0:56:13	0.5073	95.00	87.31

512	64	2:02:53	0.0763	99.88	93.09
	128	2:04:34	0.0755	99.97	93.44
768	64	2:32:29	0.0731	99.84	93.00
	128	3:48:20	0.0519	100	93.69
1,024	64	2:44:37	0.0654	99.97	92.59
	128	4:05:33	0.0410	100	93.63
2,048	64	3:42:48	0.0673	99.97	93.13
	128	6:30:00	0.0468	99.97	93.75
GPR (110)	64	1:13:44	0.1800	99.38	92.38
GPR (221)	128	5:16:05	0.1342	99.59	92.47

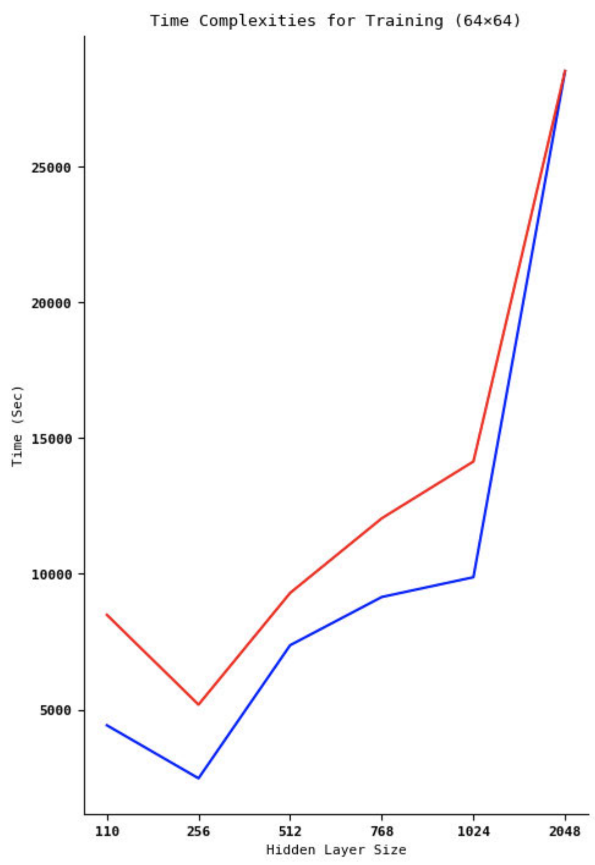
Table 19 : Observations from 12 different experiments.

1.3 Discussion

	64×64		**128×128**	
	Supplemental	**Case Study 1**	**Supplemental**	**Case Study 1**
Training	99.80	99.99	99.09	96.08
Testing	92.82	93.59	92.38	89.60

Table 20 : Comparison of averaged accuracy for training and testing for supplemental experiment and case study 1.

From Table 20, it can be seen that the difference between the accuracy for training for 64×64 has no major difference, but when used with dataset 128×128 case study 1 shows a reduction of around 3%. Similarly, for testing, when used with dataset 128×128 case study 1 exhibit similar behaviour.



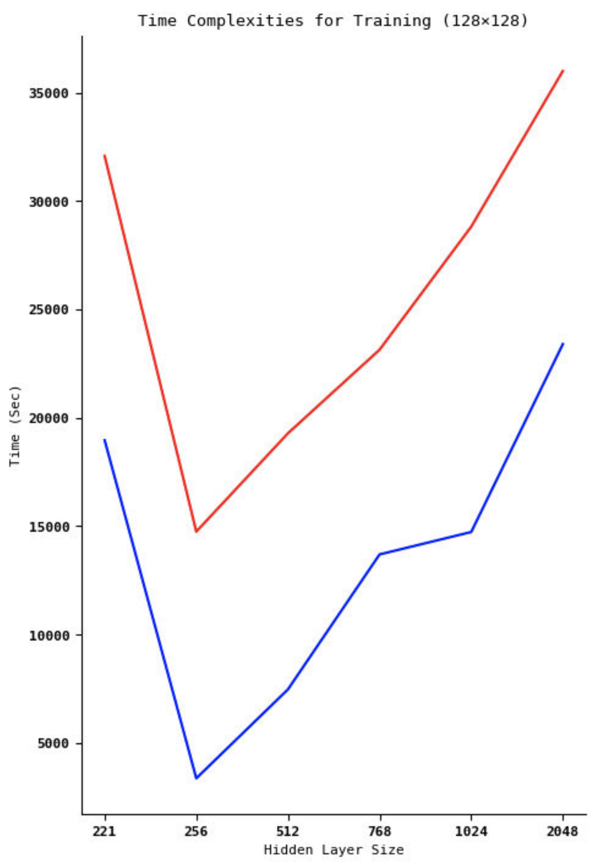


Figure 51 : Time complexities for both supplemental experiment and case study 1. The graph of the right is for dataset with 64×64 and on the left for dataset with 128×128.

By quickly analysing the figure above, it can be seen that time complexities are moving in almost the same direction for both sets of experiments. The time difference for 1,000 and 2,000 epochs for dataset with 64×64 has slight difference but dataset with 128×128 shows a big gap in time. Then, it can be concluded from these two observations that increasing the epoch will not necessarily increase the overall performance of a neural network. It may instead over generalizing and started to show the underfitting behaviour. Smaller epoch of around 1,000 should be sufficient for training.

1.4 Figures

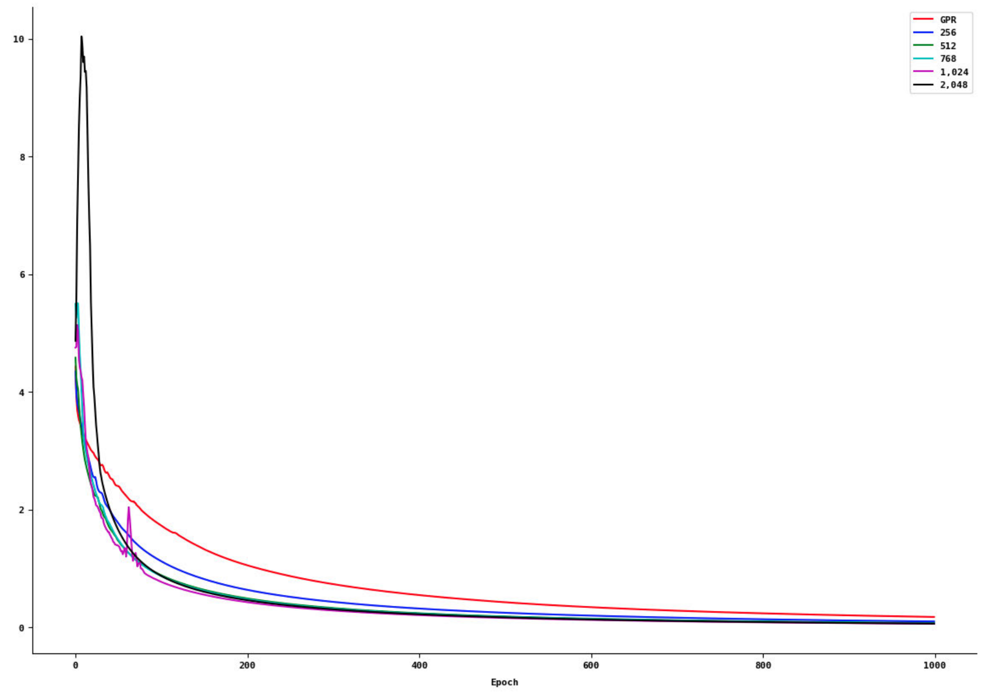


Figure 52 : Graph plotted for losses of all six trainings for 64×64 3-channel colour dataset.

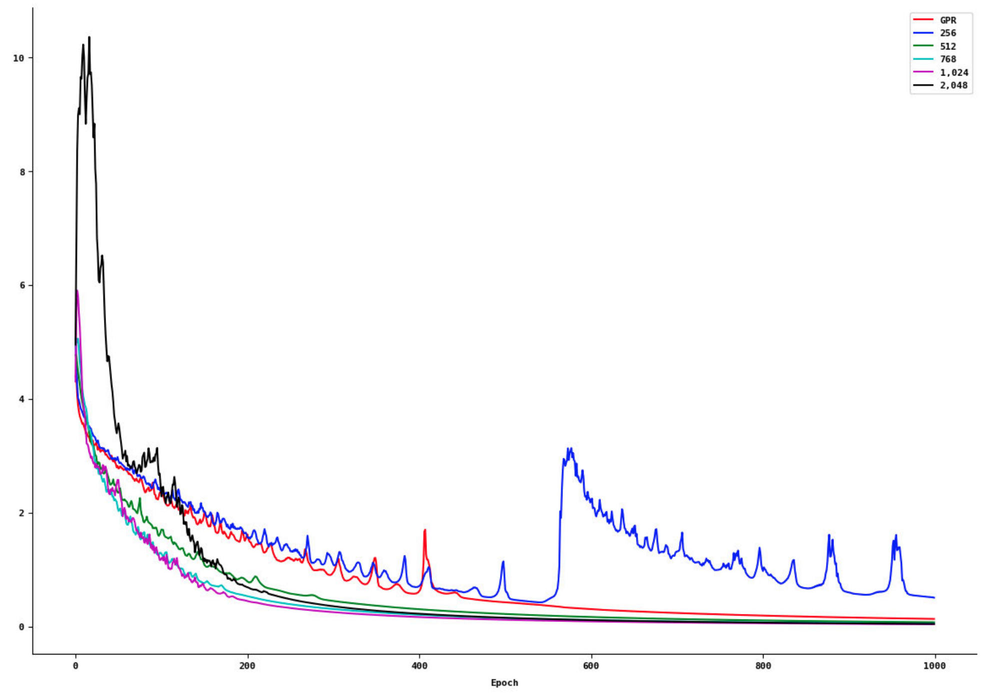


Figure 53 : Graph plotted for losses of all six trainings for 128×128 3-channel colour dataset.

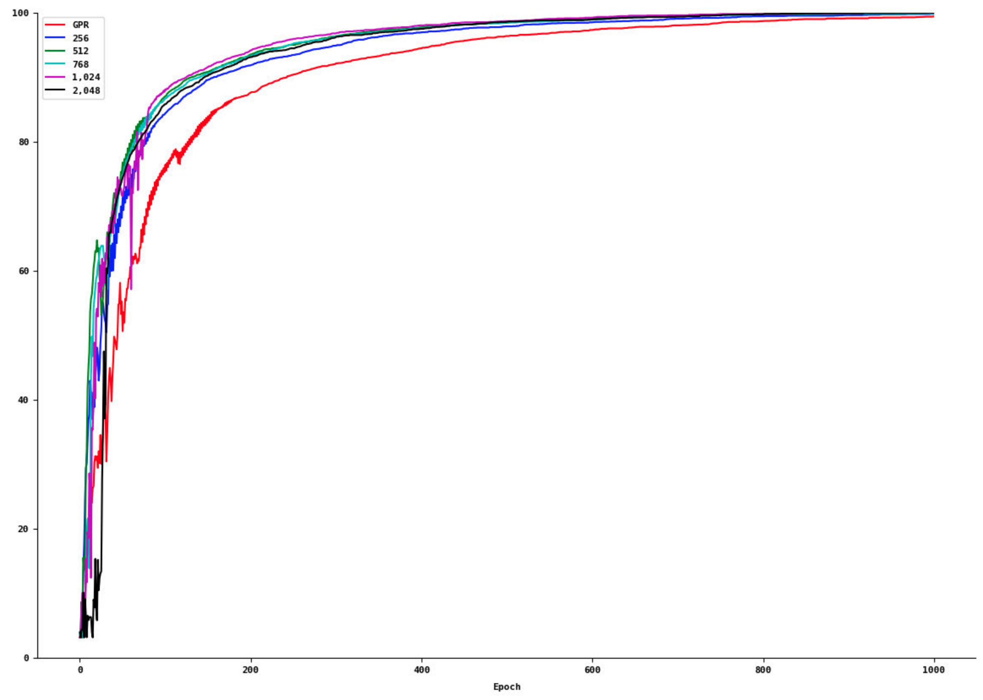


Figure 54 : Graph plotted for training accuracy of all six trainings for 64×64 3-channel colour dataset.

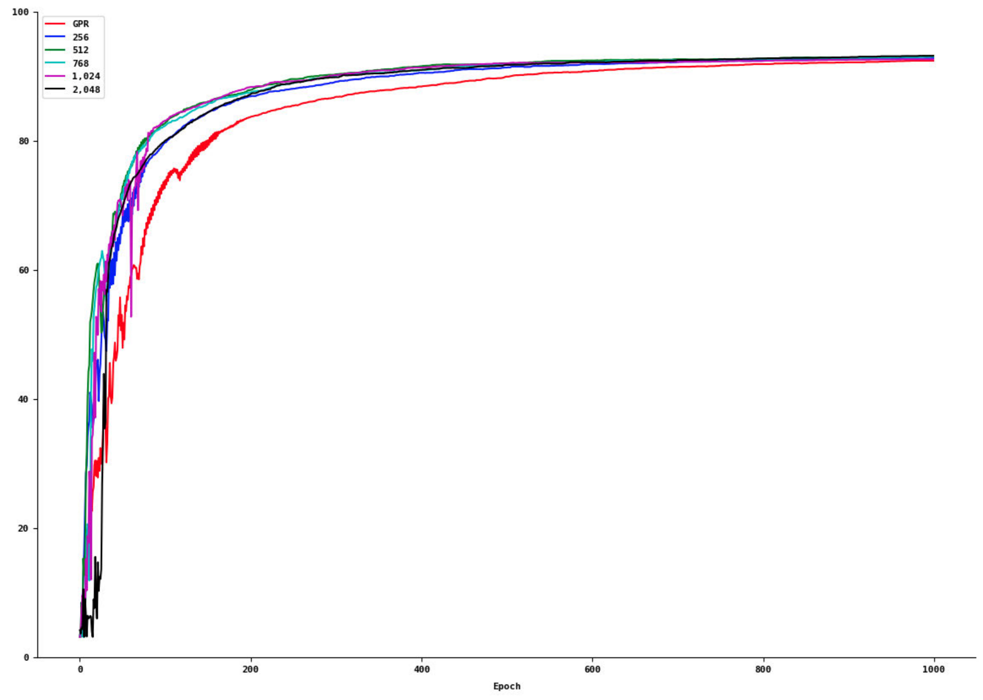


Figure 55 : Graph plotted for testing accuracy of all six trainings for 64×64 3-channel colour dataset.

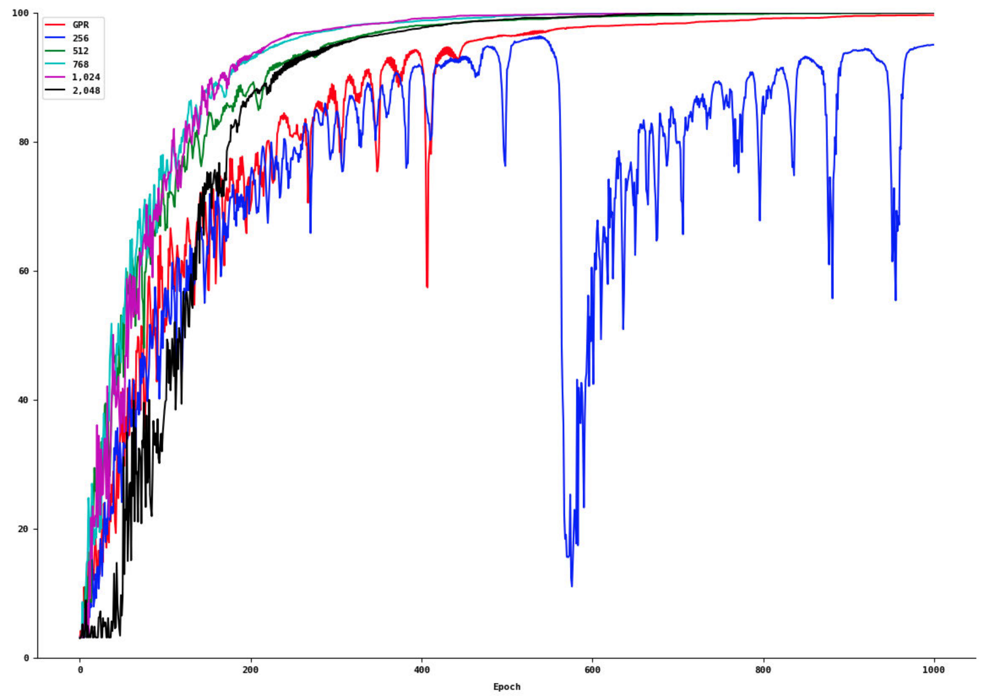


Figure 56 : Graph plotted for training accuracy of all six trainings for 128×128 3-channel colour dataset.

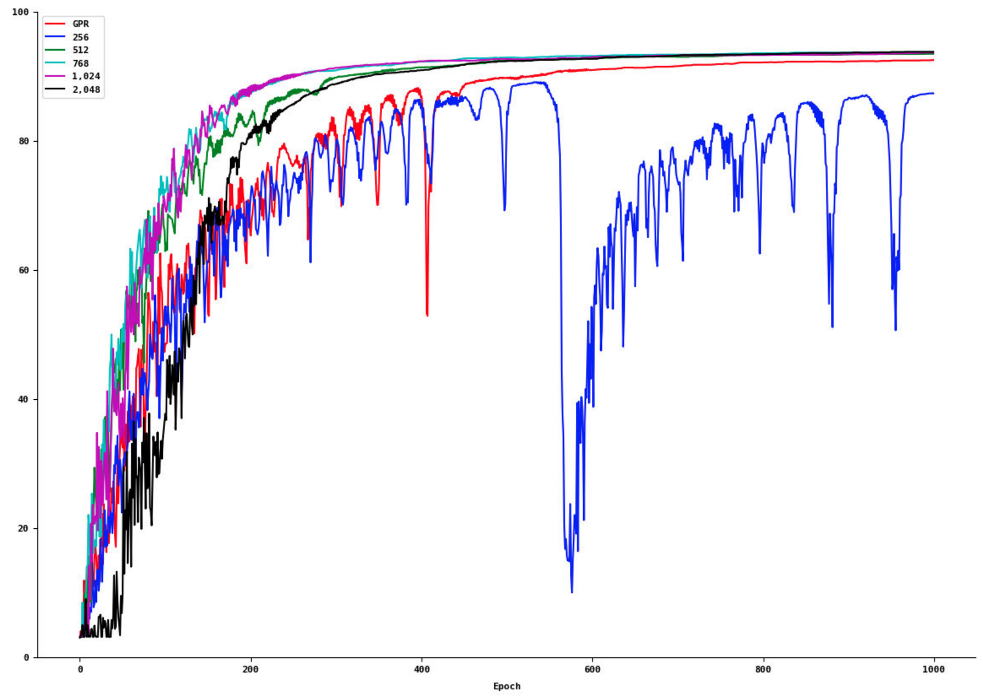


Figure 57 : Graph plotted for testing accuracy of all six trainings for 128×128 3-channel colour dataset.

2.0 FIGURES FOR CASE STUDY 1

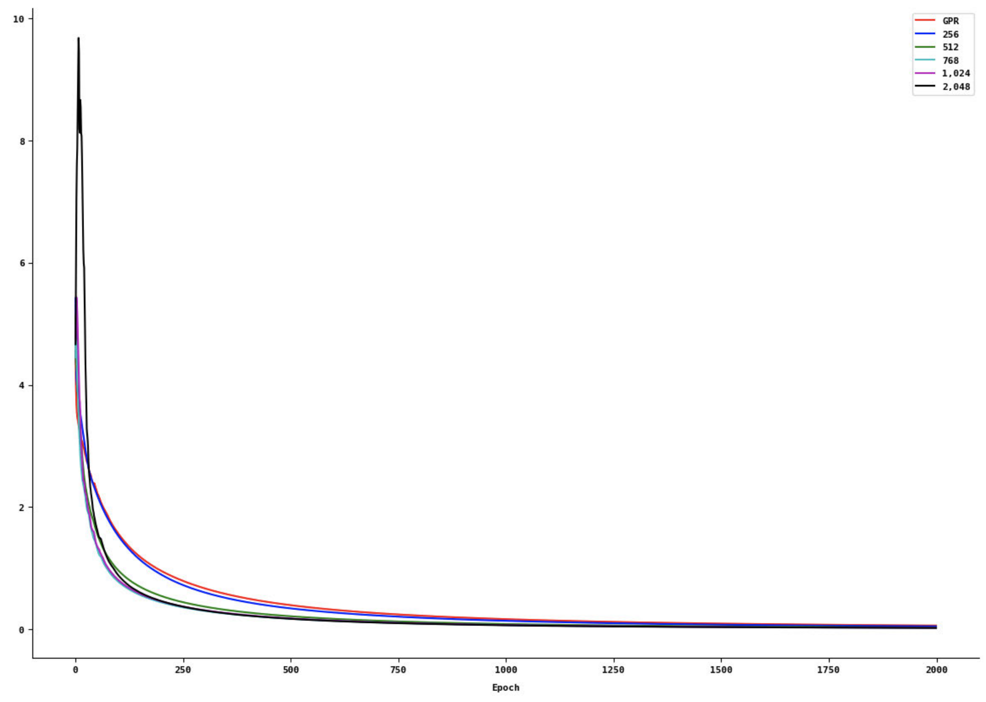


Figure 58 : Graph plotted for losses of all six trainings for 64×64 3-channel colour dataset.

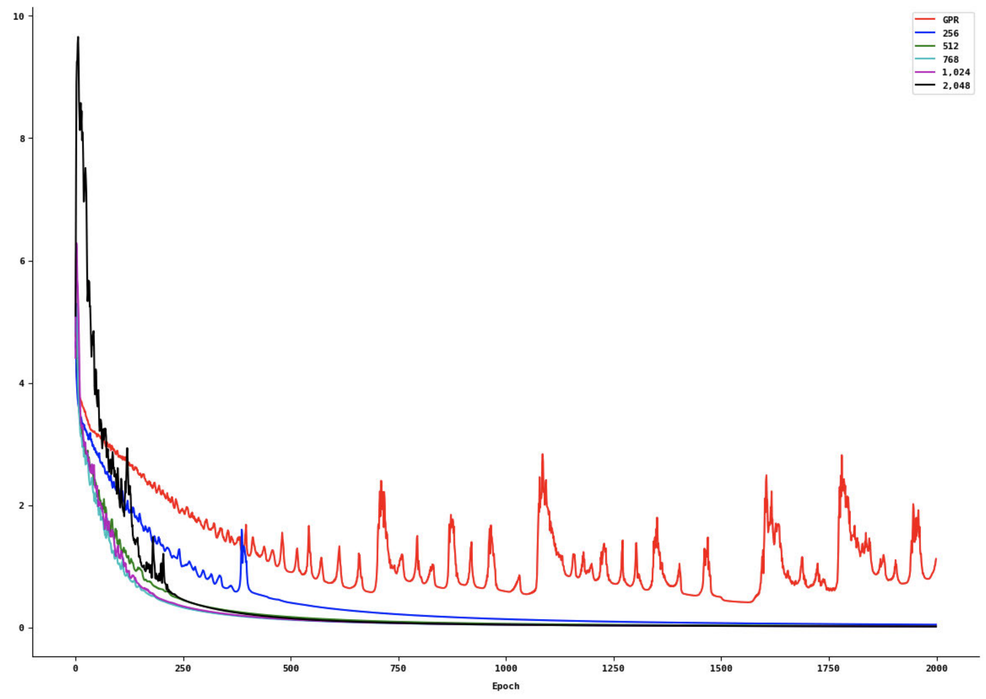


Figure 59 : Graph plotted for losses of all six trainings for 128×128 3-channel colour dataset.

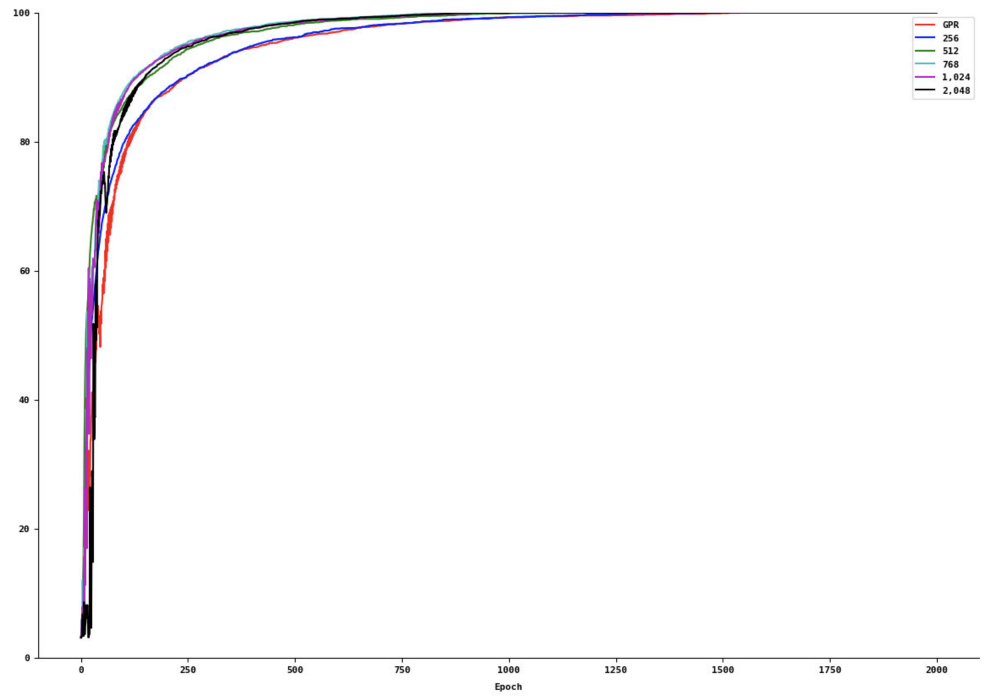


Figure 60 : Graph plotted for training accuracy of all six trainings for 64×64 3-channel colour dataset.

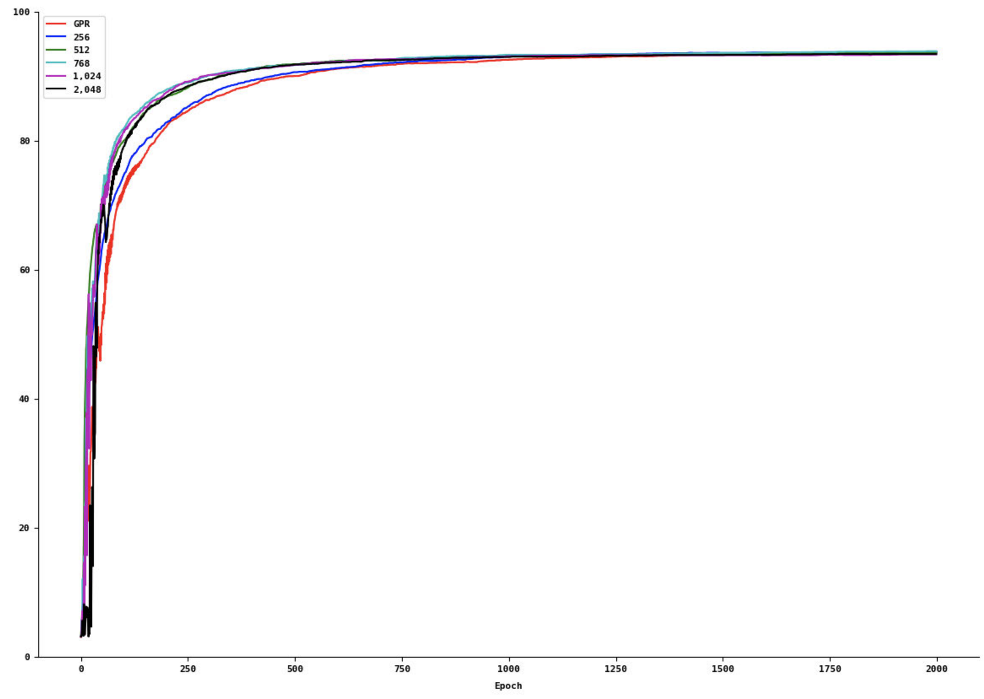


Figure 61 : Graph plotted for testing accuracy of all six trainings for 64×64 3-channel colour dataset.

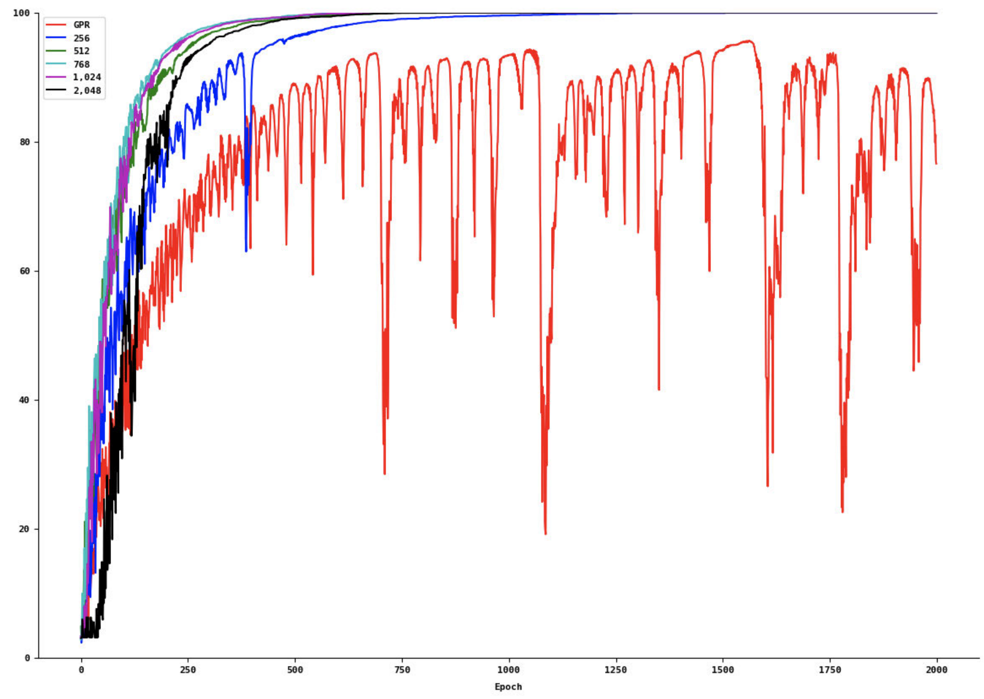


Figure 62 : Graph plotted for training accuracy of all six trainings for 128×128 3-channel colour dataset.

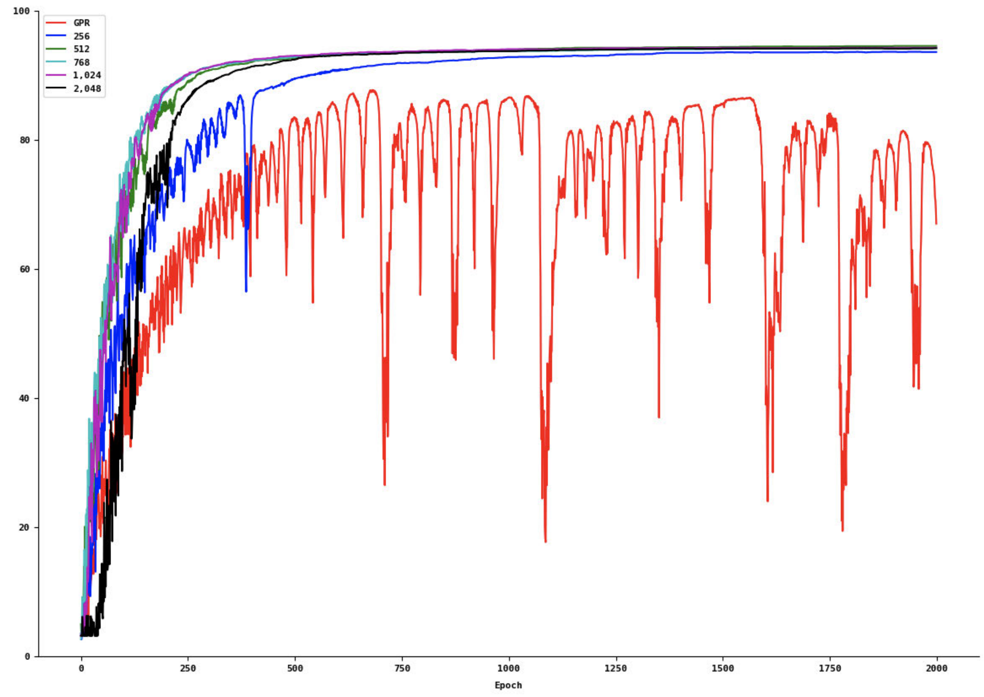


Figure 63 : Graph plotted for testing accuracy of all six trainings for 128×128 3-channel colour dataset.

3.0 FIGURES FOR CASE STUDY 2

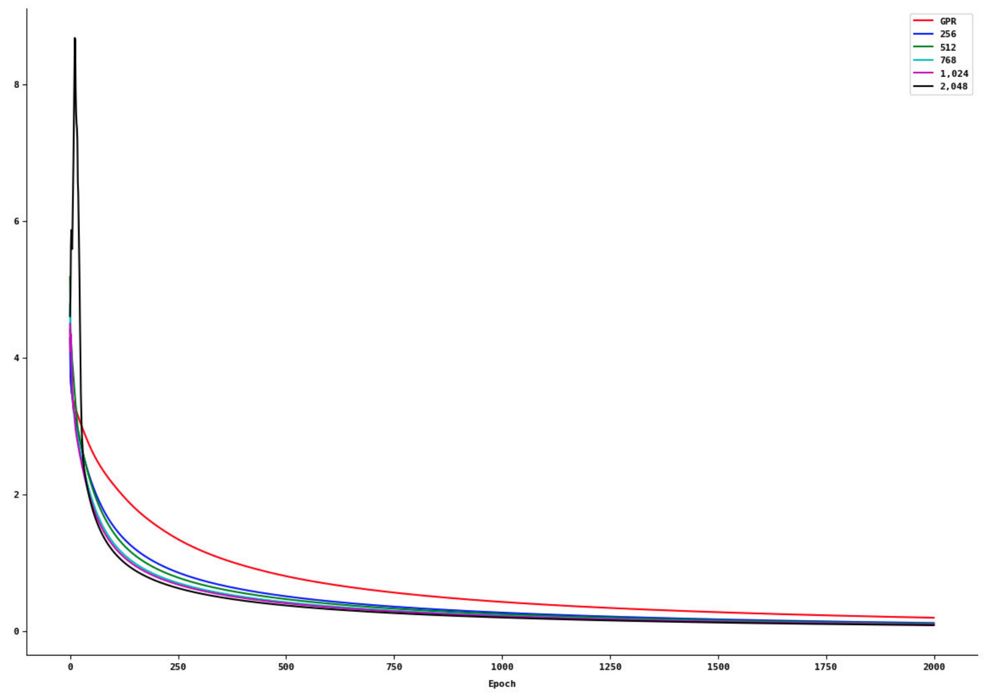


Figure 64 : Graph plotted for losses of all six trainings for 64×64 grayscale dataset.

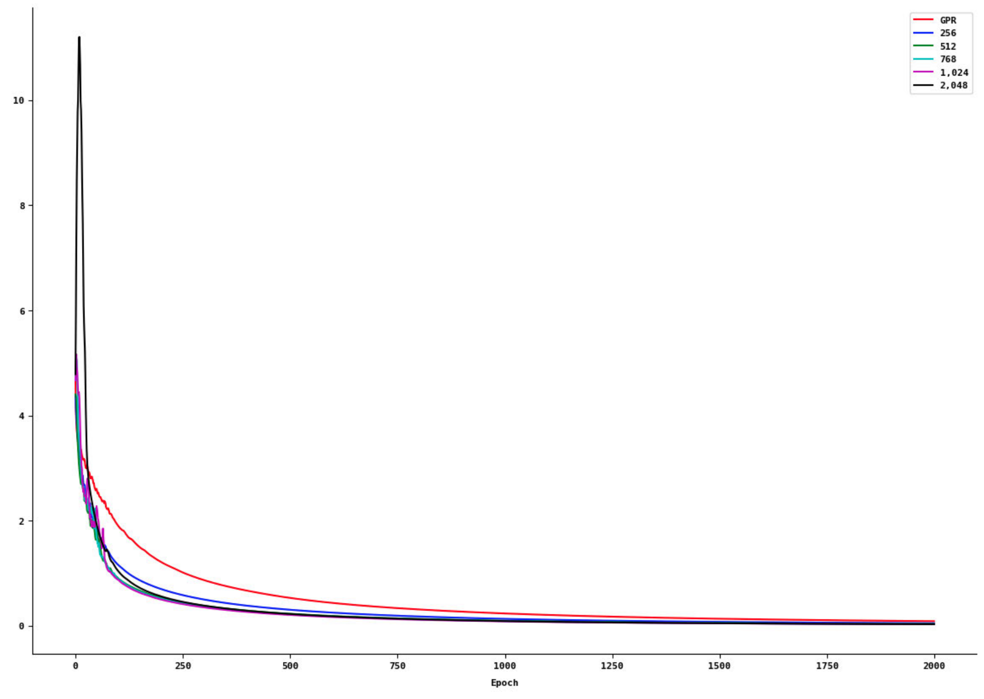


Figure 65 : Graph plotted for losses of all six trainings for 128×128 grayscale dataset.

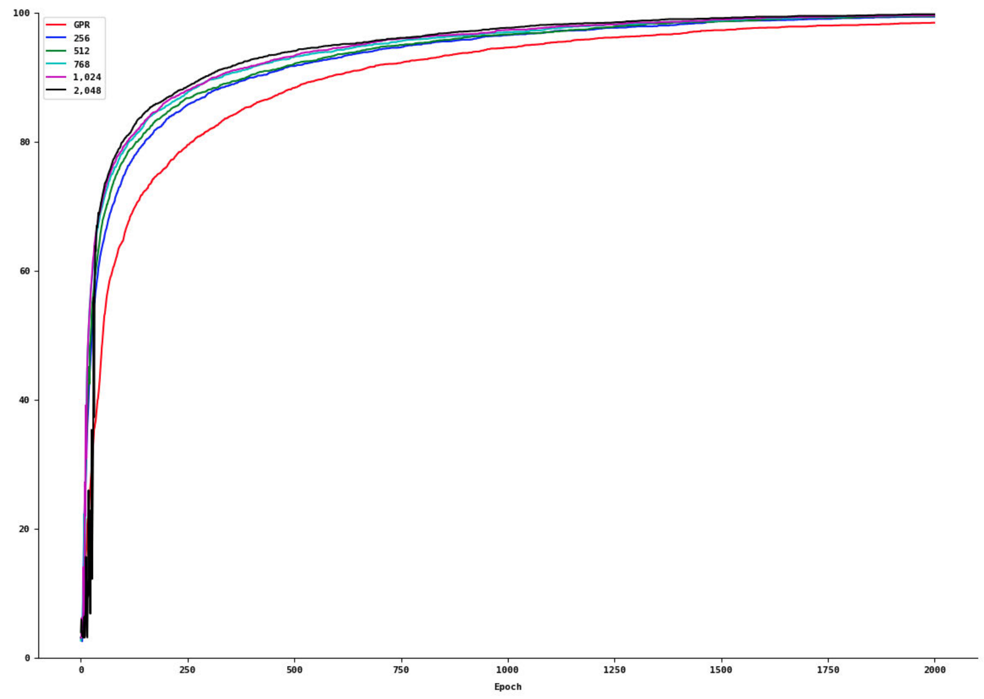


Figure 66 : Graph plotted for training accuracy of all six trainings for 64×64 grayscale dataset.

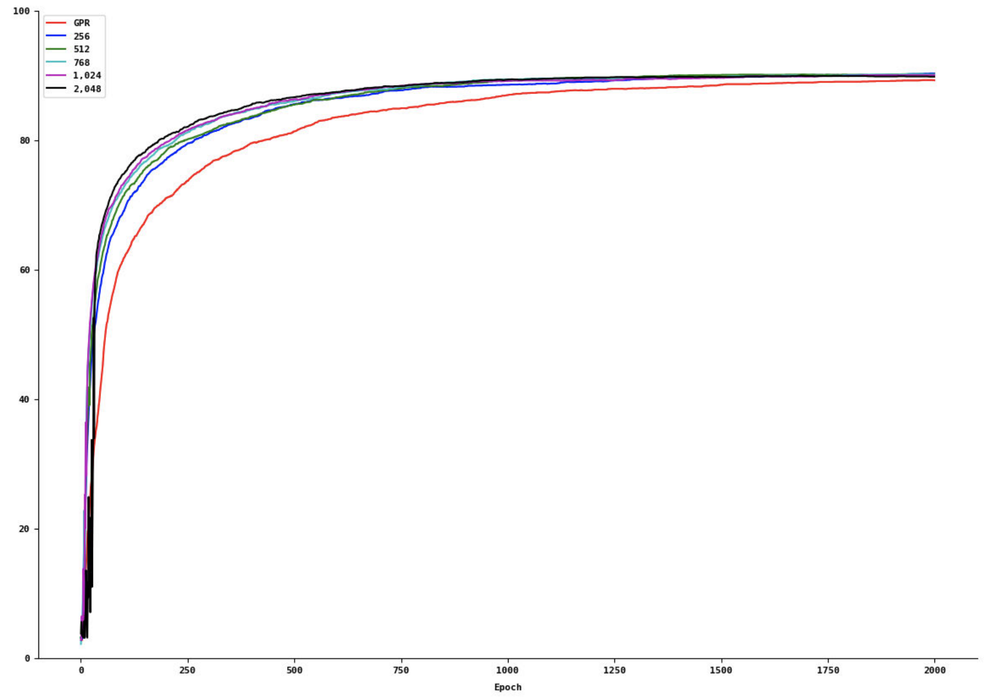


Figure 67 : Graph plotted for testing accuracy of all six trainings for 64×64 grayscale dataset.

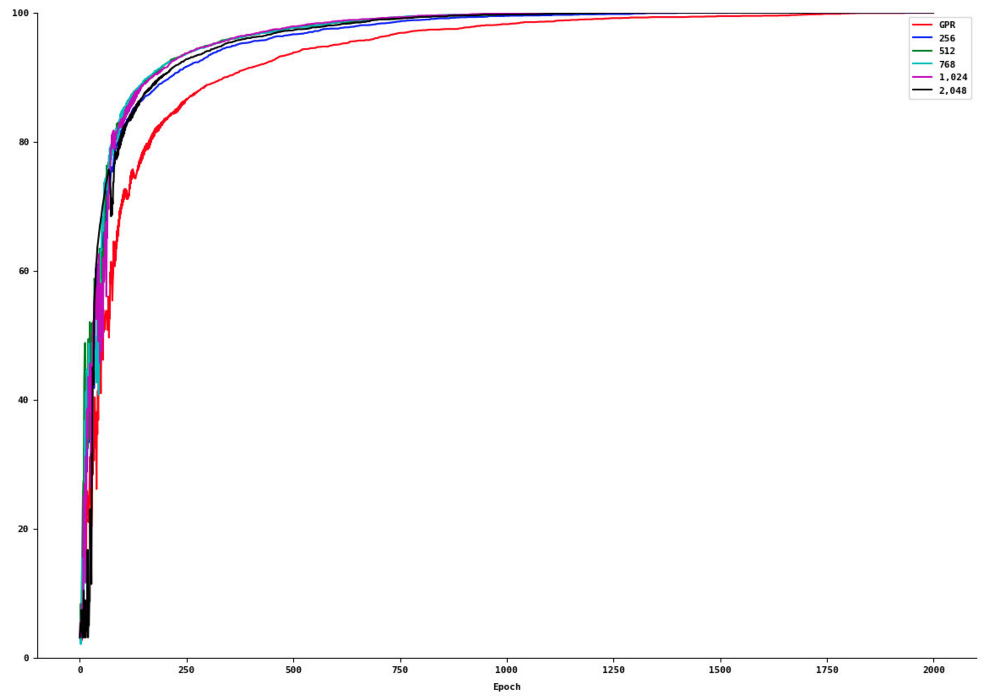


Figure 68 : Graph plotted for training accuracy of all six trainings for 128×128 grayscale dataset.

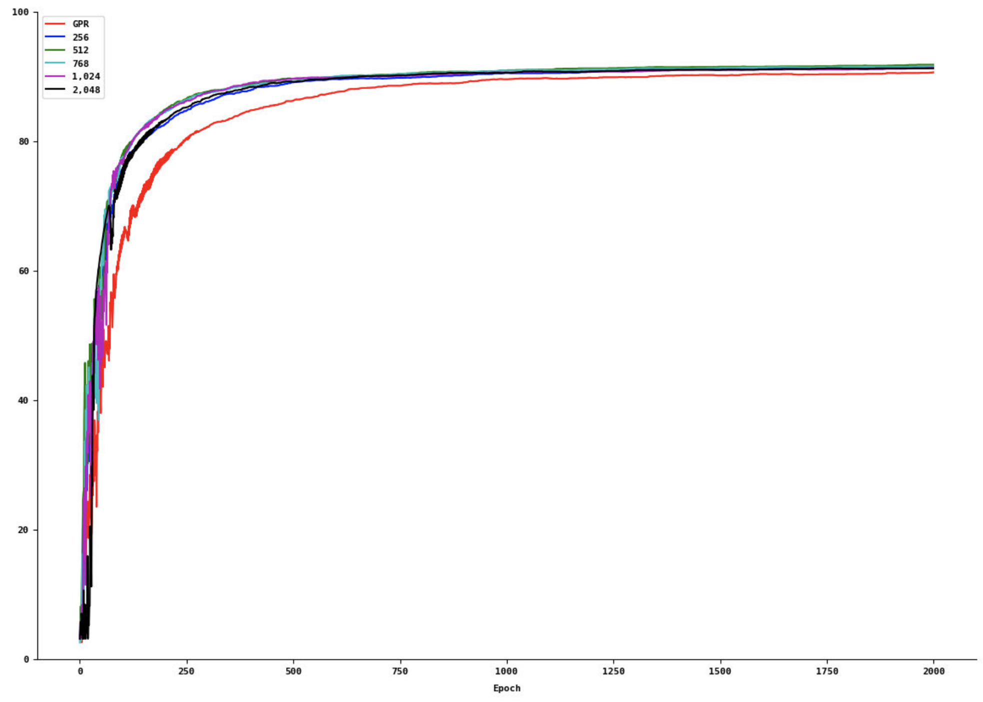


Figure 69 : Graph plotted for testing accuracy of all six trainings for 128×128 grayscale dataset.

4.0 FIGURES FOR CASE STUDY 3

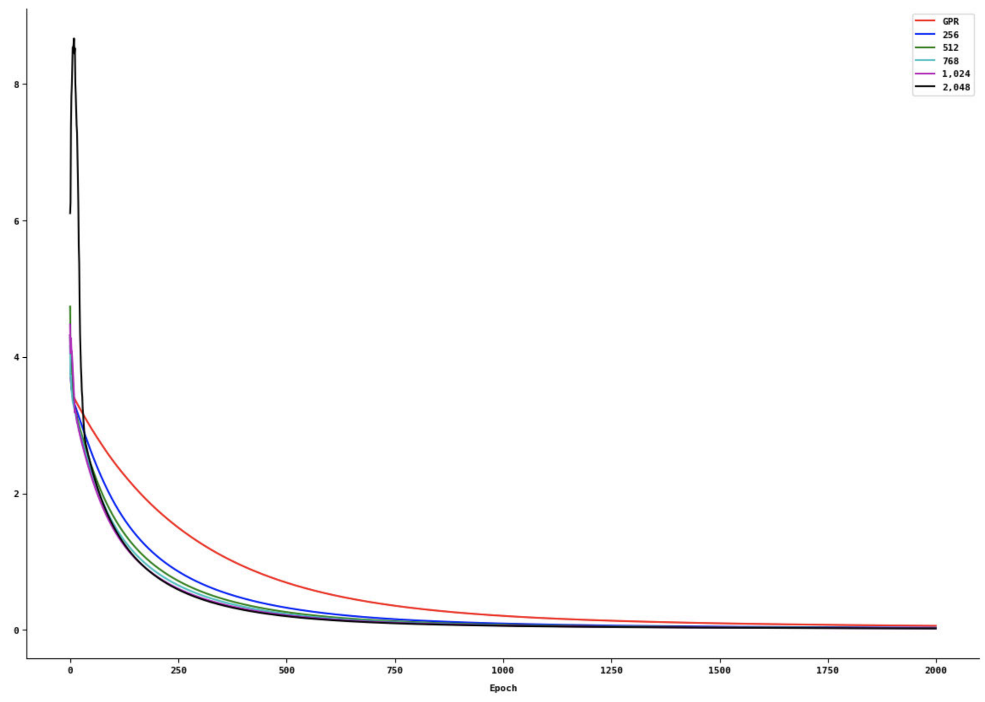


Figure 70 : Graph plotted for losses of all six trainings for 64×64 LBP dataset.

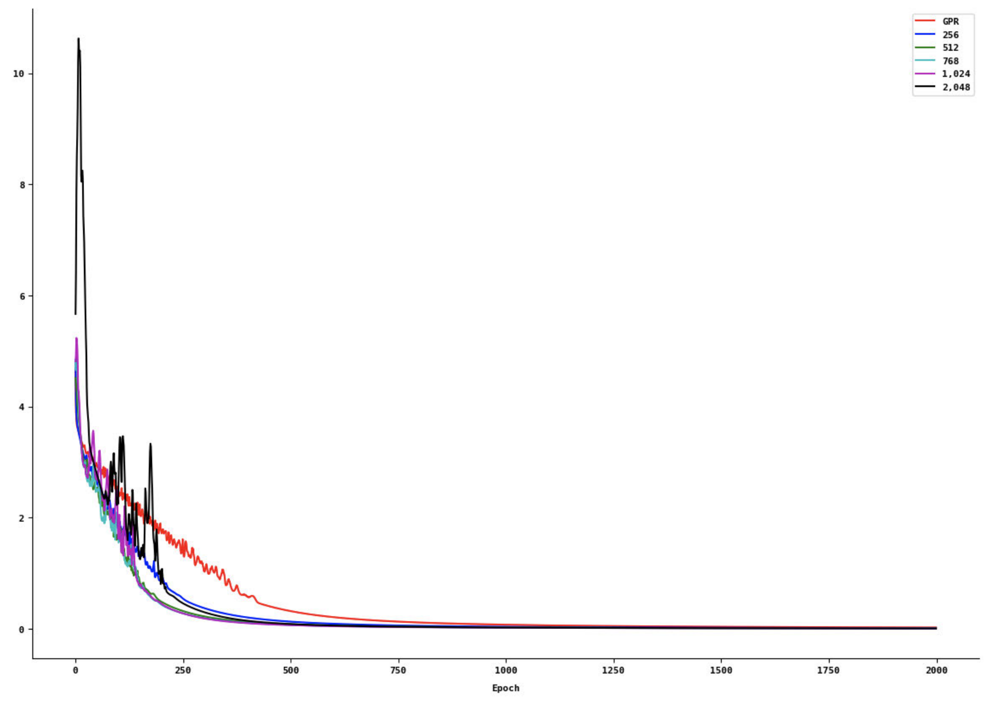


Figure 71 : Graph plotted for losses of all six trainings for 128×128 LBP dataset.

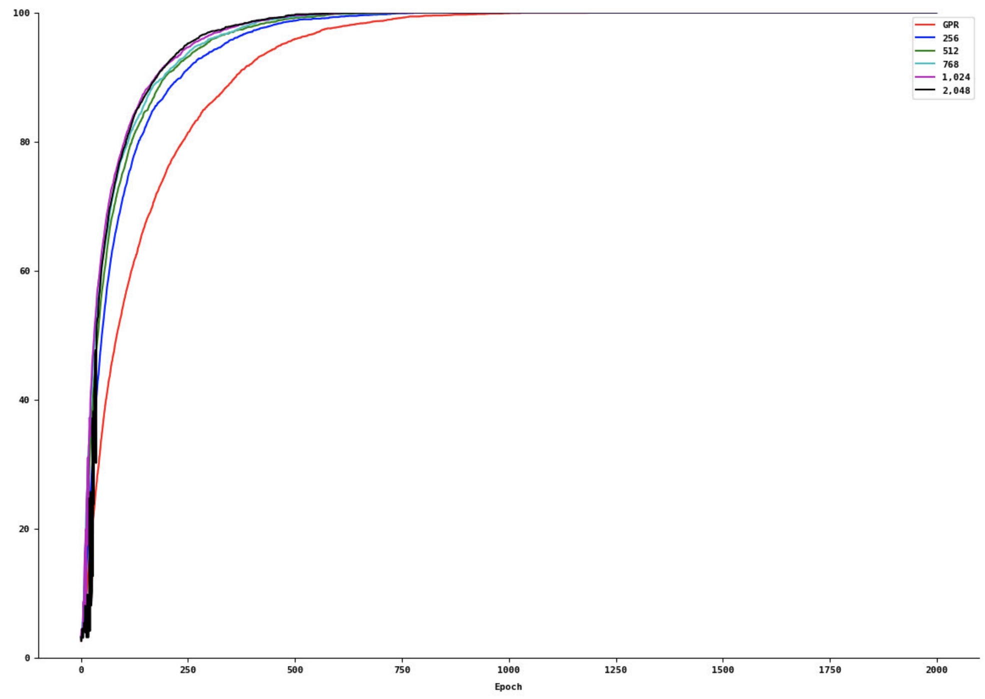


Figure 72 : Graph plotted for training accuracies of all six trainings for 64×64 LBP dataset.

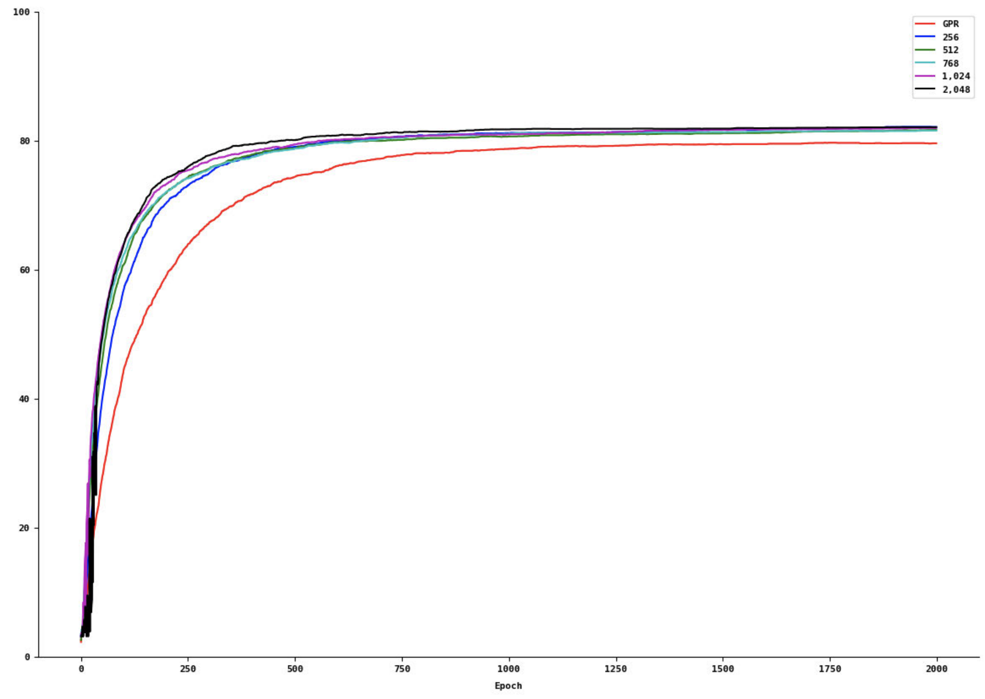


Figure 73 : Graph plotted for testing accuracies of all six trainings for 64×64 LBP dataset.

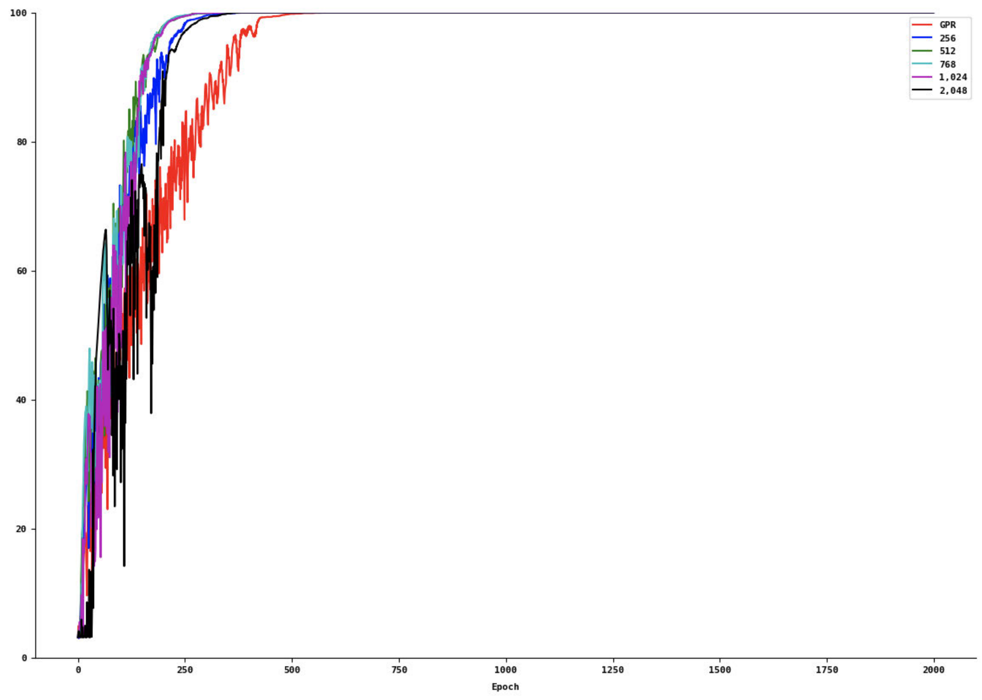


Figure 74 : Graph plotted for training accuracies of all six trainings for 128×128 LBP dataset.

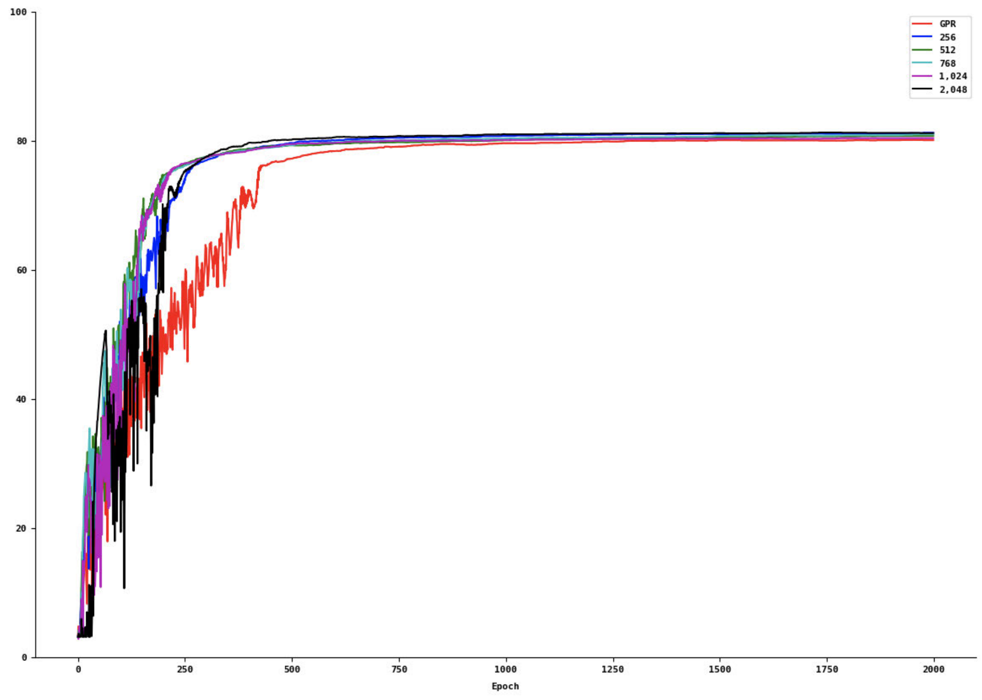


Figure 75 : Graph plotted for training accuracies of all six trainings for 128×128 LBP dataset.

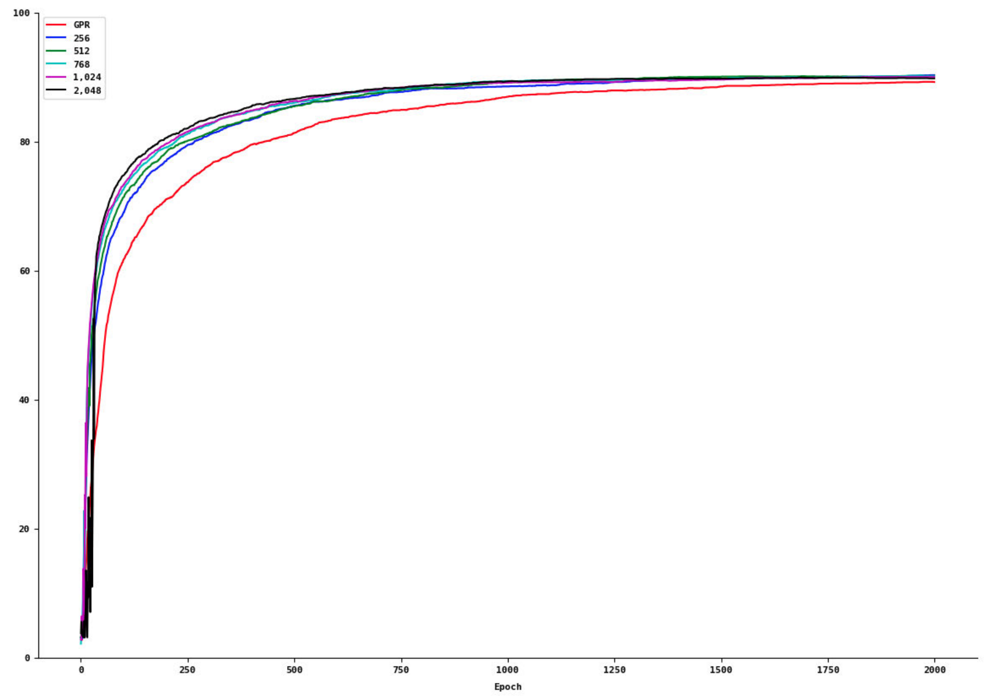


Figure 76 : Graph plotted for testing accuracy of all six trainings for 64×64 grayscale dataset.

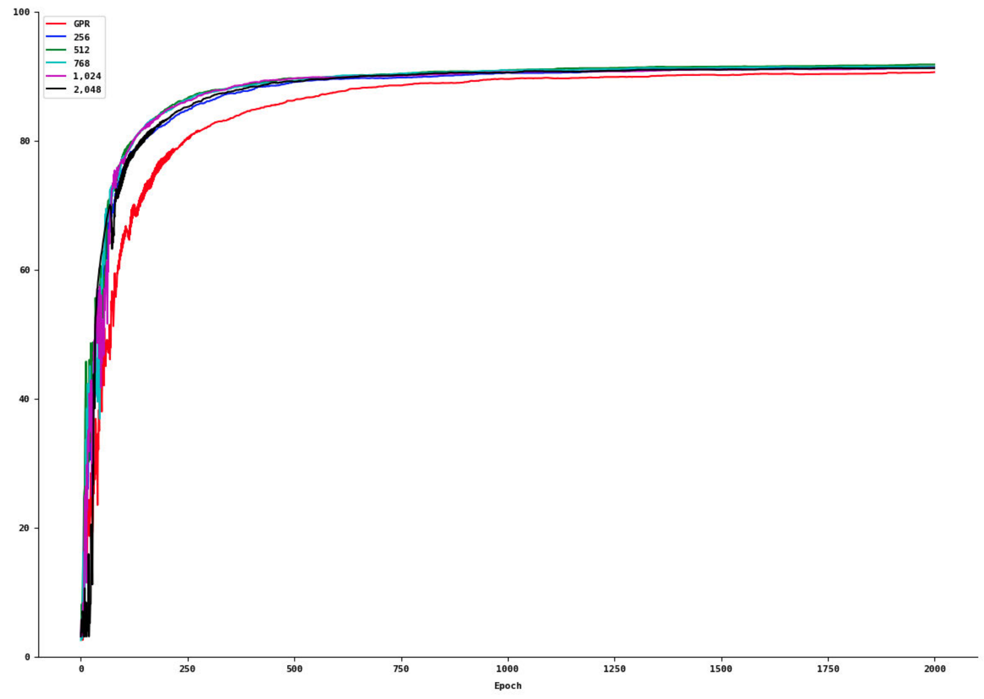


Figure 77 : Graph plotted for testing accuracy of all six trainings for 128×128 grayscale dataset.

5.0 FIGURES FOR CASE STUDY 4

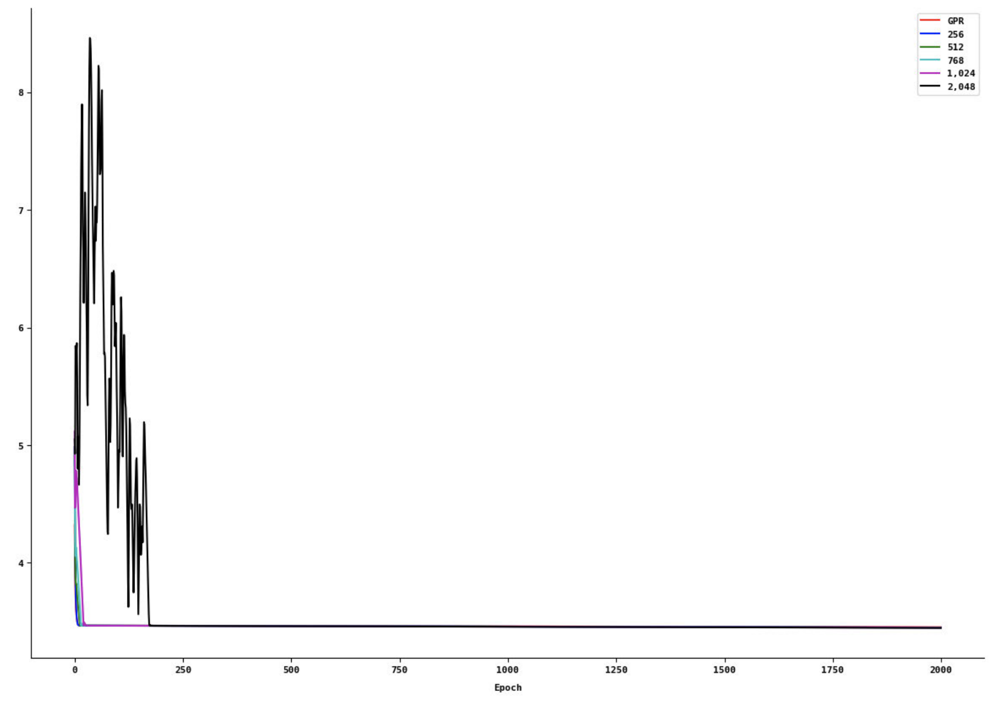


Figure 78 : Graph plotted for losses of all six trainings for 64×64 LBPH dataset.

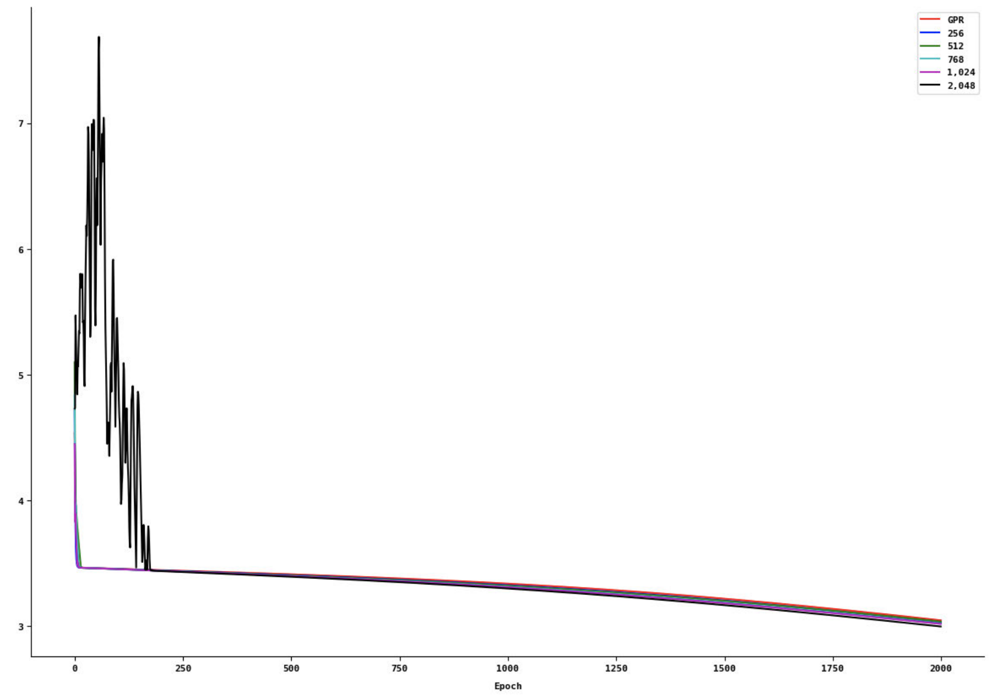


Figure 79 : Graph plotted for losses of all six trainings for 128×128 LBPH dataset.

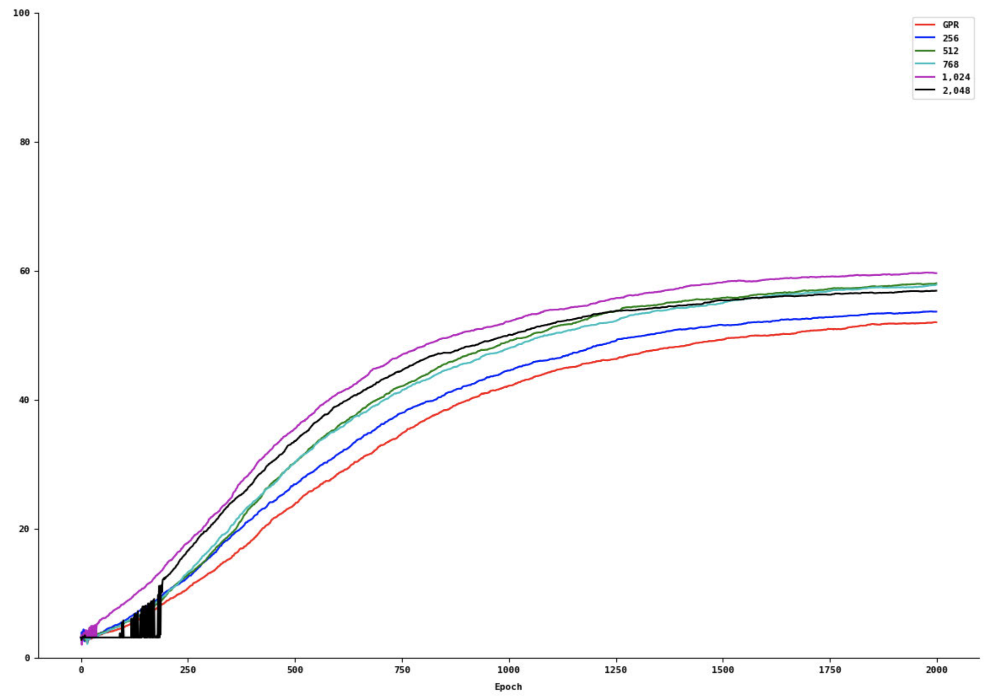


Figure 80 : Graph plotted for training accuracies of all six trainings for 64×64 LBPH dataset.

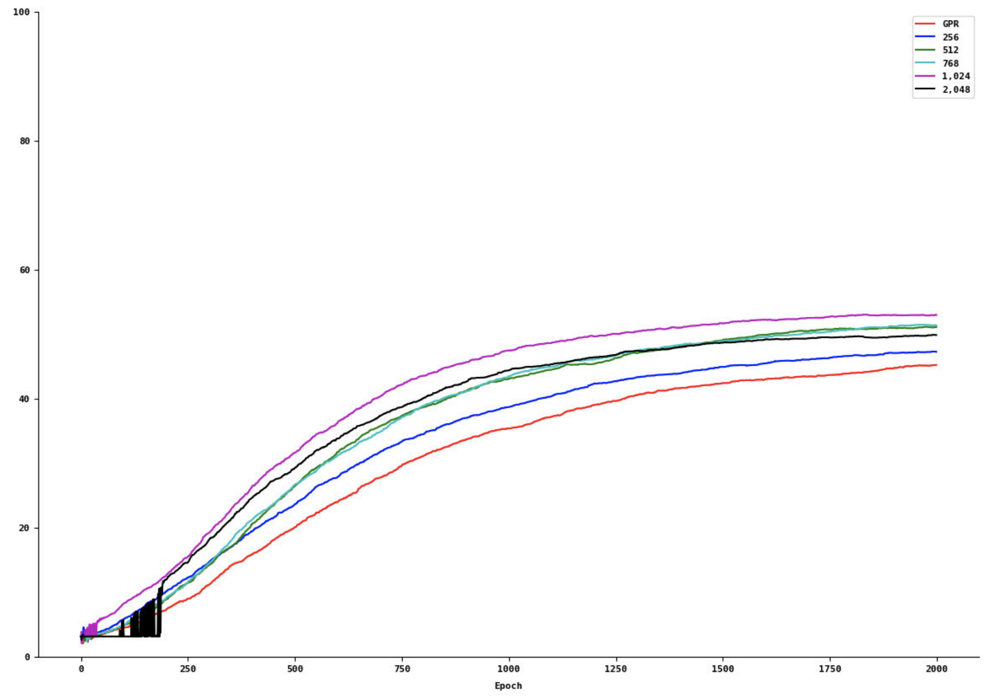


Figure 81 : Graph plotted for testing accuracies of all six trainings for 64×64 LBPH dataset.

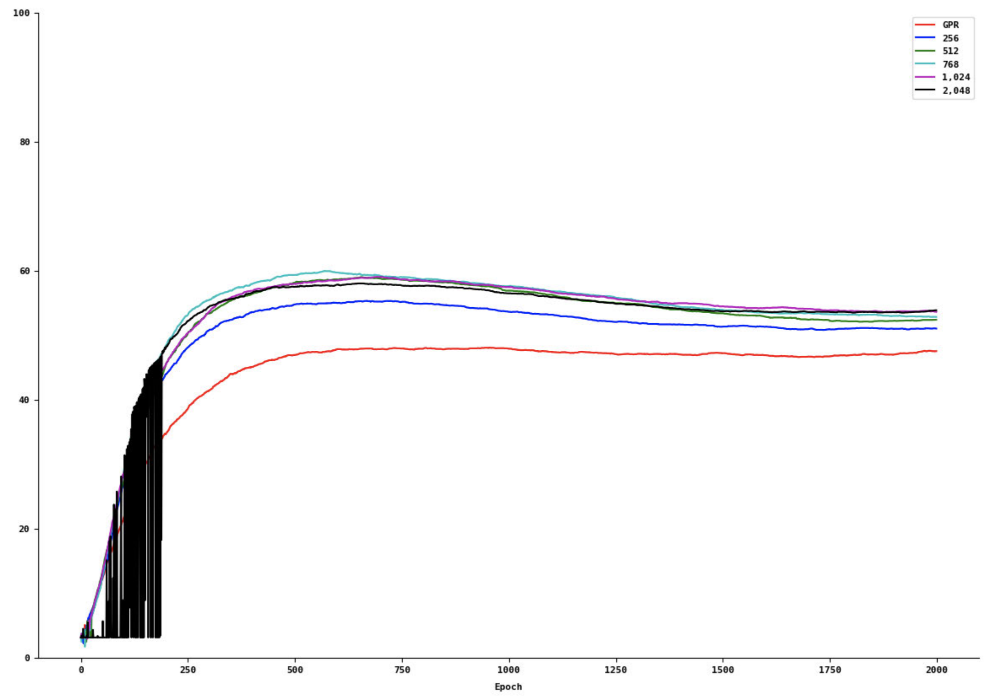


Figure 82 : Graph plotted for training accuracies of all six trainings for 128×128 LBPH dataset.

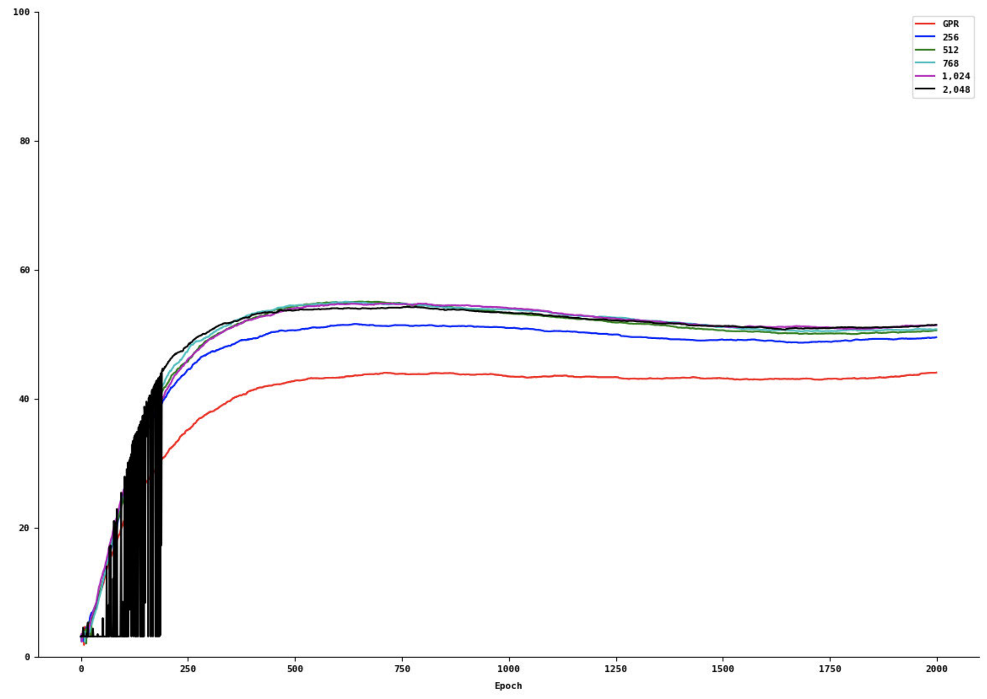


Figure 83 : Graph plotted for testing accuracies of all six trainings for 128×128 LBPH dataset.

APPENDIX

Ⓑ

Analysis using Confusion Matrix

Two experiments have been chosen to be discussed in term of confusion matrix. They are colour and grayscale, both with images in 128×128 dimension. The experiments have variables fixed and there were 768 neurons in the hidden layer. Both of them are considered balance in terms of used number of images for training and testing. The performance of a classification model on a set of test data when the true values are known can be analysed using confusion matrix table. However, this evaluation has not been used in concluding the most applicable experiment due to the lack of discussion in the theory; without making this report a thick text book. However, it has been chosen to be discussed in a small appendix as a base to permit future research.

1.0 CONFUSION MATRIX (COLOUR DATASET)

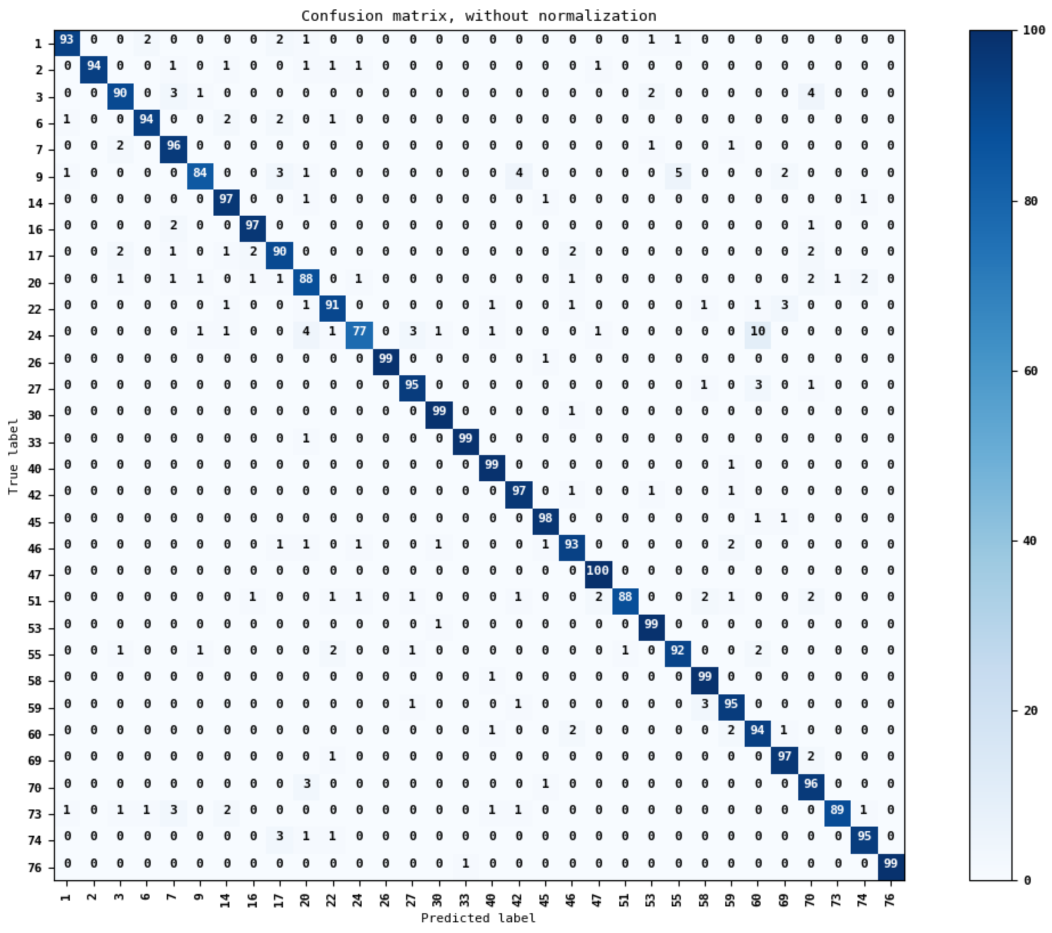


Figure 84 : A confusion matrix with heat-map plotted for case study 1. The image dimension in the dataset is 128×128 and experimented with 768-neurons in the hidden layer.

In the figure above, it can be seen that the identification accuracy was quite high, except for 6 volunteers. Volunteer number 24 has the lowest identification accuracy and the volunteer number 47 has as high as 100% identification. Volunteers that have been identified the most for false-negative are 17, 27, 42, 55, 60 and 70.

2.0 CONFUSION MATRIX (GRAYSCALE DATASET)

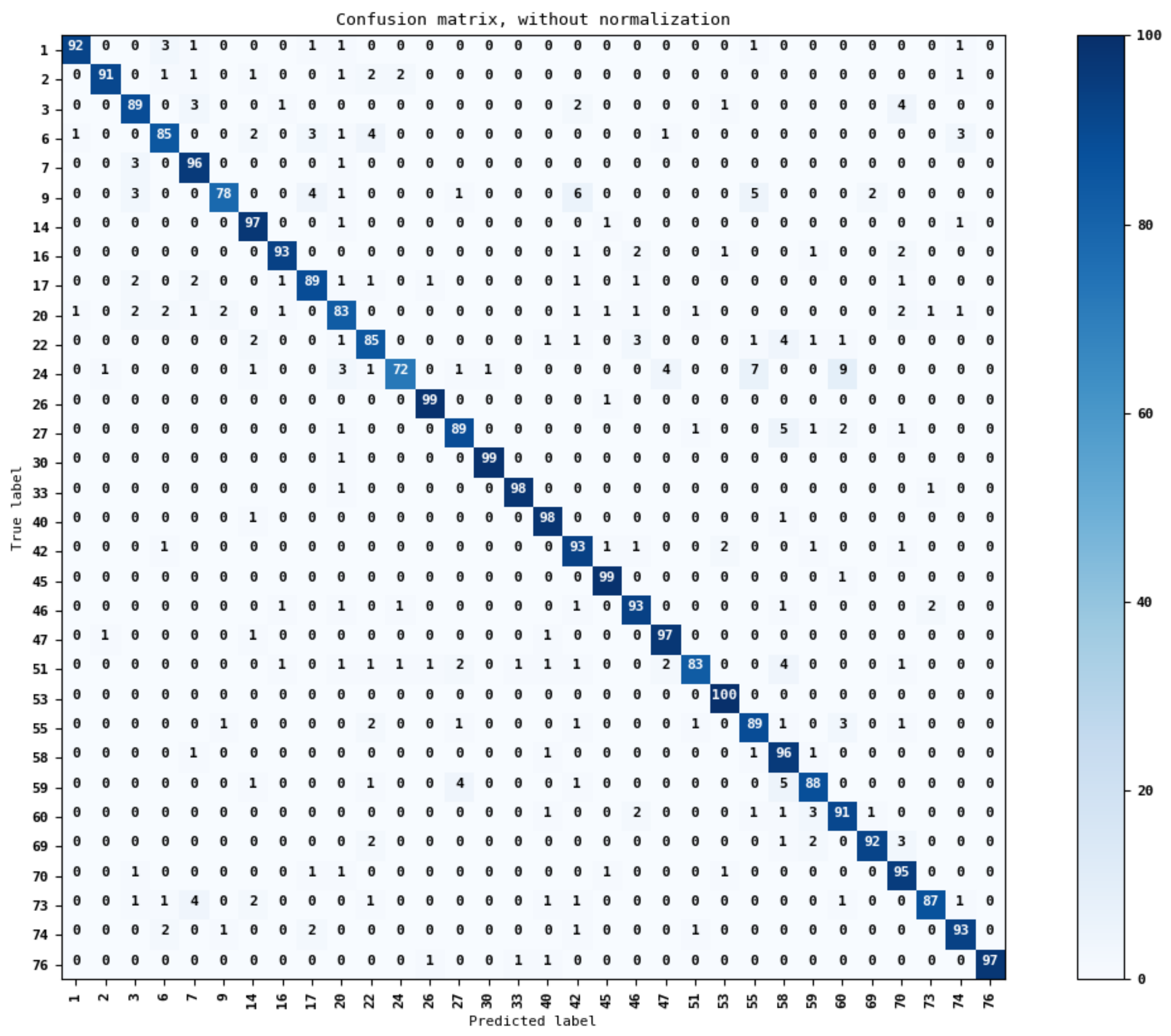


Figure 85 : A confusion matrix with heat-map plotted for case study 2. The image dimension in the dataset is 128×128 and experimented with 768-neurons in the hidden layer.

From the figure above, it can be seen the same volunteer number 24 having the lowest identification. But the identification with the highest identification is 53. Volunteers that have been identified the most for false-negative are 20, 42, 55, 60 and 70.

3.0 IDENTIFICATION TEST FOR IDENTIFIED VOLUNTEERS

Identification tests are made based on the identified false-negatives from two sections above. The test is using the prediction process using saved *trained model* from experiment in case study 1 and case study 2. The input are images that were taken during similar session for images that were used for training and testing. They are volunteer number 17 *(no permission given for use in this report)*, 27, 42, 55, 60 and 70. Two additional tests will be for 24, 47 and 53 for volunteer identified with the highest and lowest true-positives. Those with false-negatives are volunteer who are being wrongly identified as the identity for another volunteer.

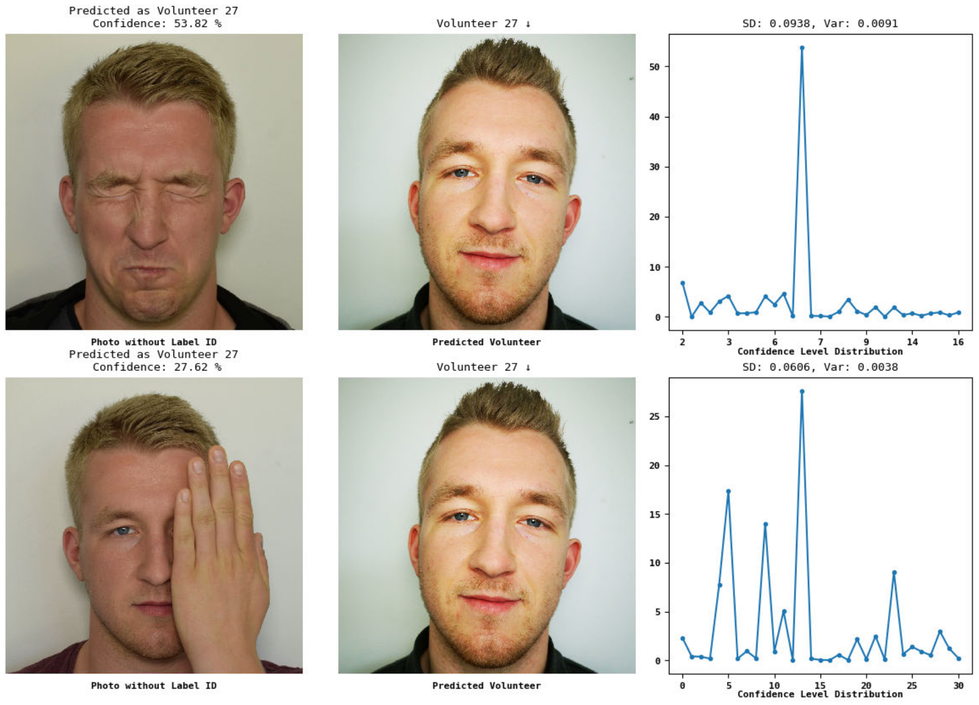


Figure 86 : Two images of volunteer 27 chosen for the test; one non-natural facial expression and one with occlusion. Both images have been successfully identified to him. Despite the successes, the confidence levels are quite low. The confidence levels for the image with occlusion are uneven, with three other high spikes are noticeable.

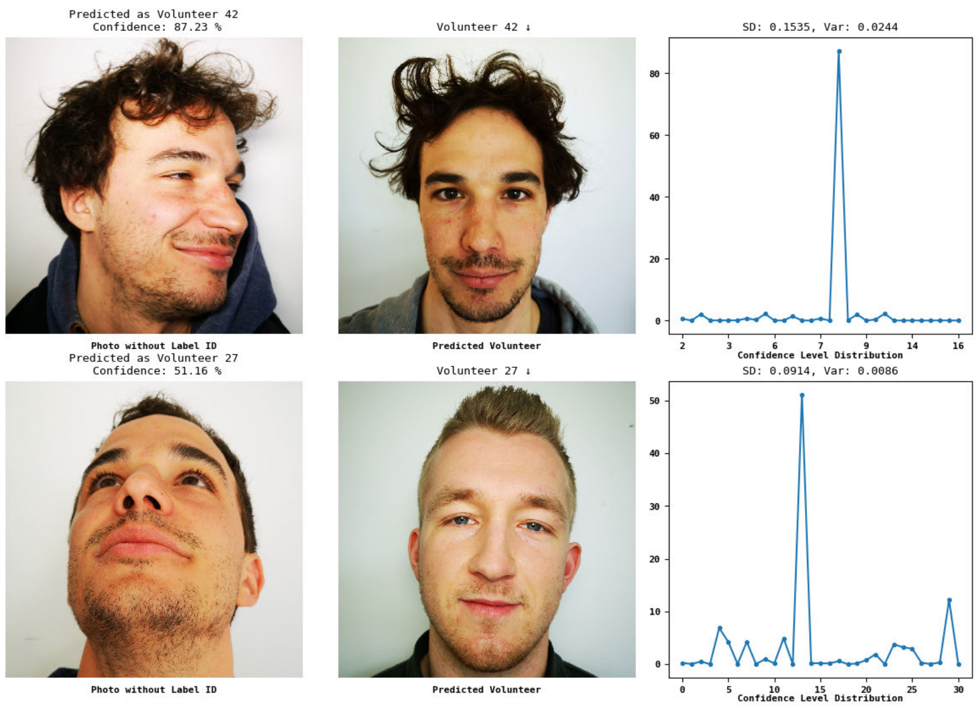


Figure 87 : Two images of volunteer 42 have been tested, and despite being the occurrence for false-positive, the test shows only one success with 87% despite not facing forward. The second has a confidence level of 51% but was identified as volunteer 27. Looking at the distribution graph, distributions are uneven.

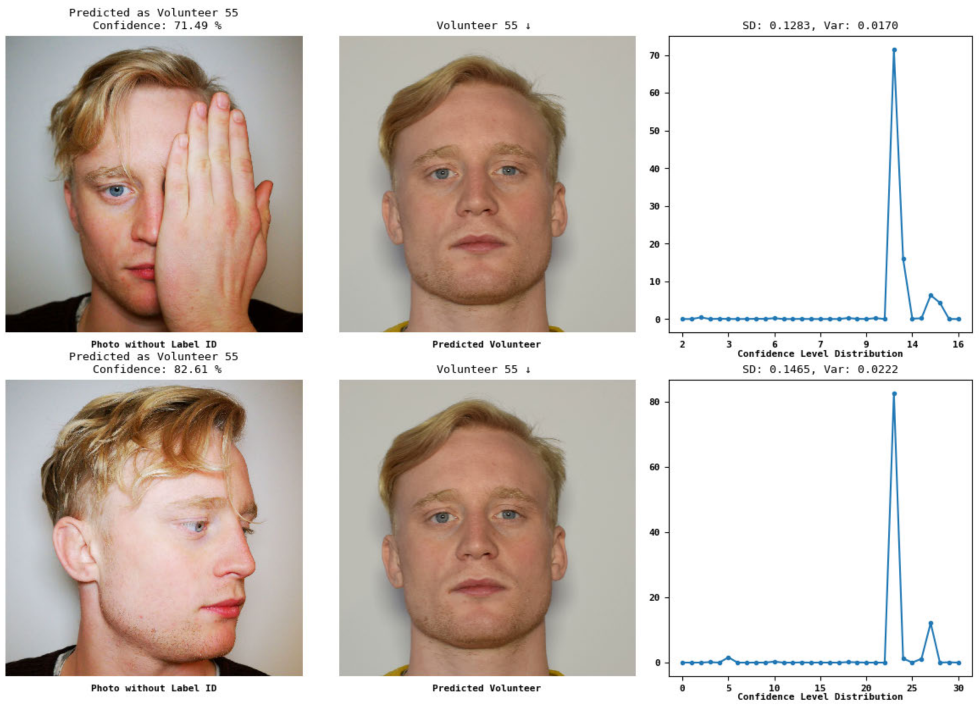


Figure 88 : Two images for volunteer 55 were chosen with one occlusion and one facing to the side. Both have been successfully identified. Despite the occlusion, the confidence level was quite high at 71%.

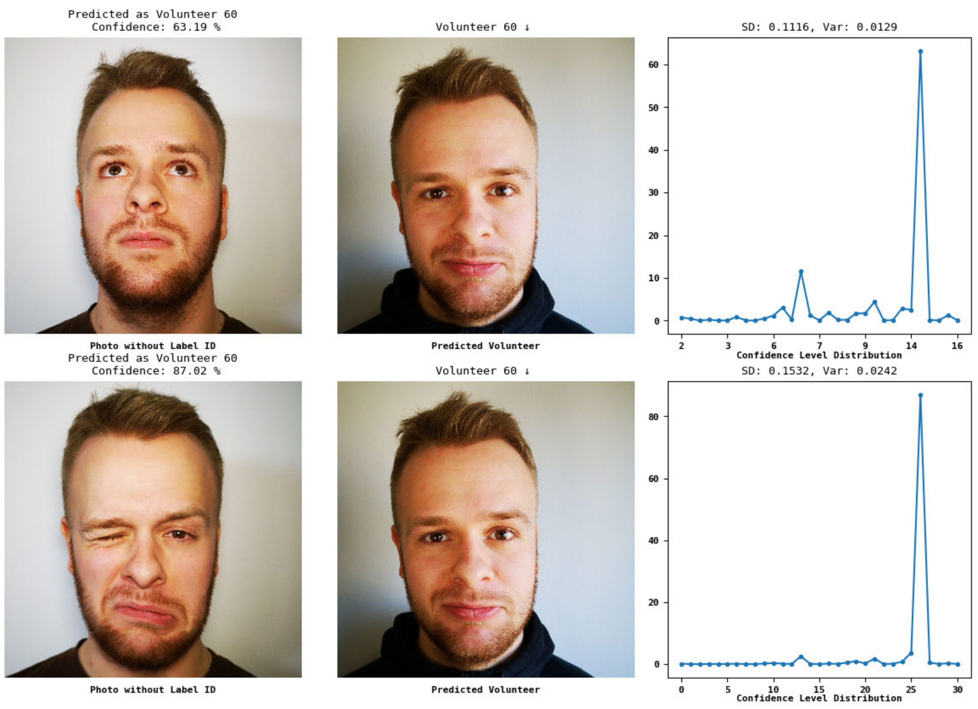


Figure 89 : Two images for volunteer 55 were chosen; one looking upward and one with non-natural facial expression. Both have been successfully identified, despite both test images came from similar sessions with very minor change in the face angle, the difference for confidence level is 25%.

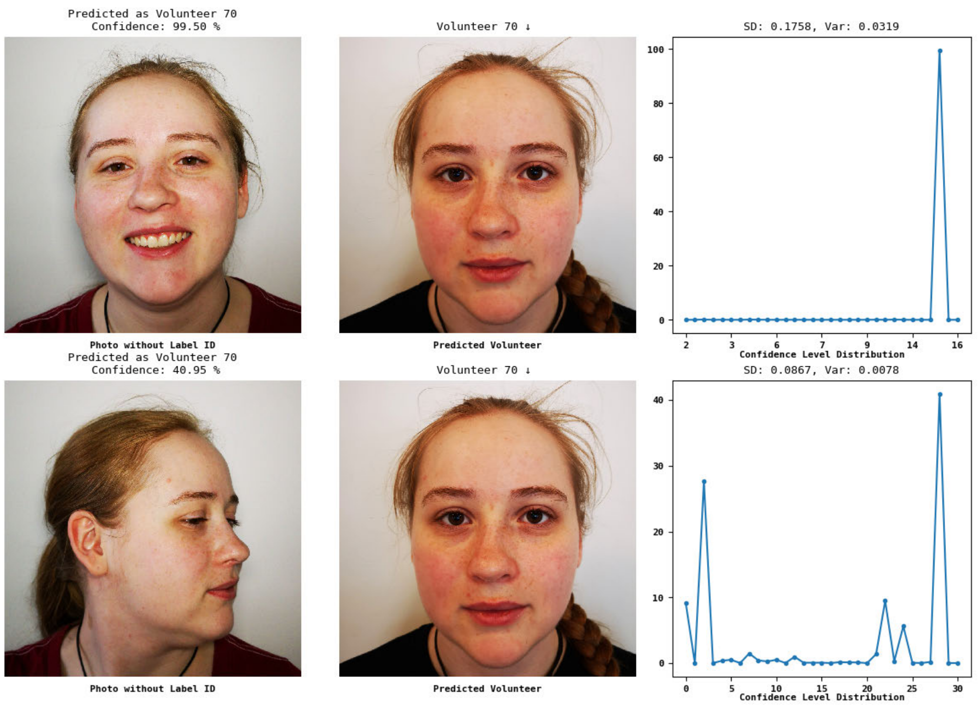


Figure 90 : Two images for volunteer 70 were chosen and tested and both have been successfully identified. The first one with 99% confidence level and one facing to the side has 41%.

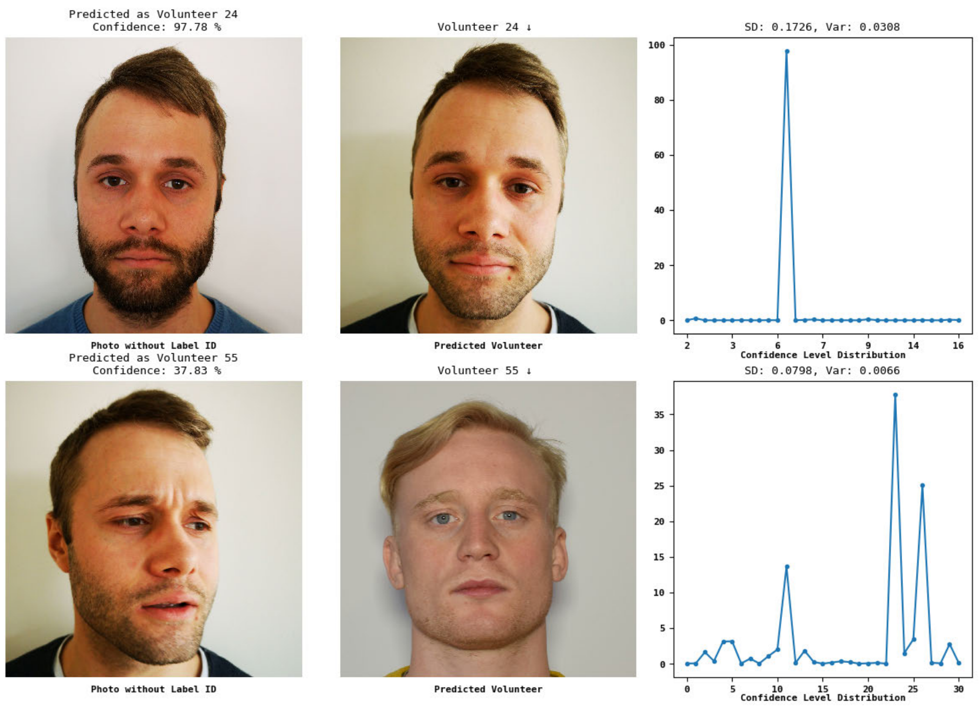


Figure 91 : Volunteer number 24 has the lowest true-positive in both experiments. Two images of him were chosen, one with beard and one facing to the side. The test gives one true-positive and one false-negative. The one with beard has an identification of 98% but the other one was identified as volunteer 55. However, it the confidence level is quite low and the distribution seems to be very uneven.

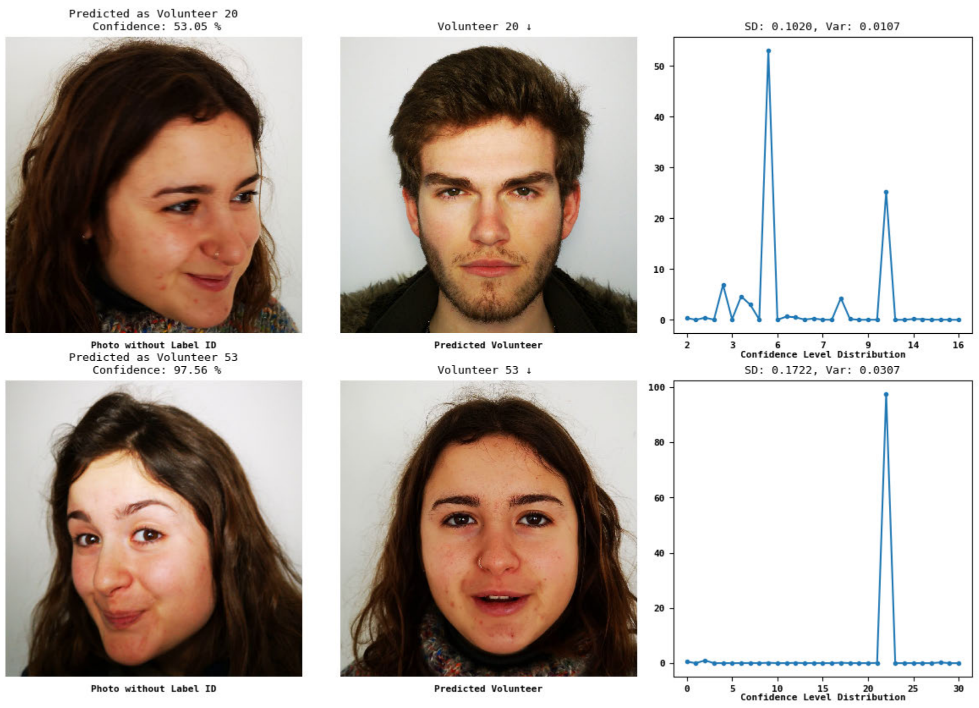


Figure 92 : Volunteer 53 the highest true-positive in the grayscale experiment. Two images for volunteer 53 have been chosen. However, the test only gives one true-positive and one false-negative. In terms of confidence level, the false-positive is 53% and true-positive is 98%.

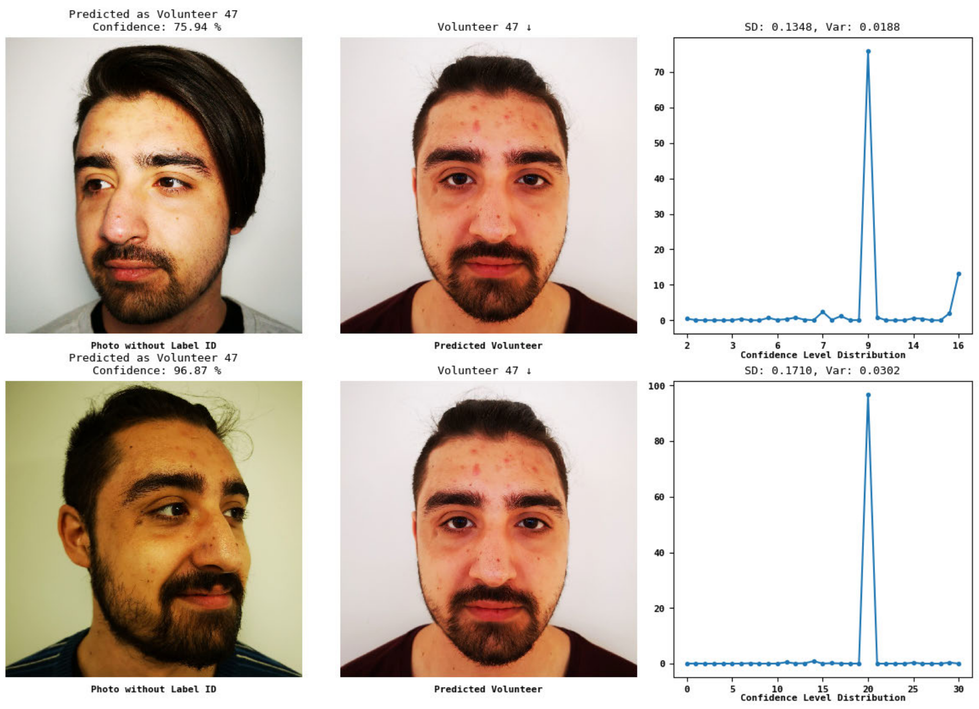


Figure 93 : Volunteer number 47 has the highest accuracy score in the experiment with colour dataset. His identification remains high in the experiment with grayscale dataset. When tested, both of them give true-positive for his identity. The classification distribution looks quite consistent.

4.0 CONCLUSION

In concluding and choosing the most well performed experiment, confusion matrix can be used to provide a sound argument. It can also be used to see which identity can be tested further. In the repeated test, three prominent identities were identified; they are 47, 55 and 73. Further research can focus on studying more about prominent identities and also how confusion matrix can be used to come up with the threshold for confidence level.

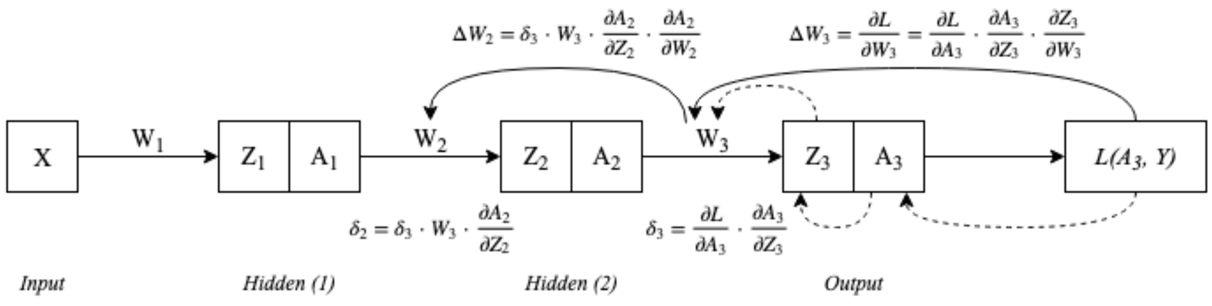
APPENDIX

Ⓒ

Delta Rule for Backpropagation

DELTA RULE, δ, FOR BACK PROPAGATION

Delta Rule starts by calculating the error residuals, L, which is the difference between the actual outputs, $\hat{Y}$, and the target outputs, Y. Using this error value, connecting weights are then increased or decreased in proportion to the error multiplied by a scaling factor, α. The complex part of this learning mechanism is for the network to determine which input contributed the most to an incorrect output and how does that element get changed to reduce the error. The process is backward process which started from the output layer and then move towards hidden layers before reaching the input layer in a linear sequence.



$\Delta W_2 = \delta_3 \cdot W_3 \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial A_2}{\partial W_2}$
$\Delta W_3 = \frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial W_3}$
X
W_1
Z_1
A_1
W_2
Z_2
A_2
W_3
Z_3
A_3
$L(A_3, Y)$
$\delta_2 = \delta_3 \cdot W_3 \cdot \frac{\partial A_2}{\partial Z_2}$
$\delta_3 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3}$
Input
Hidden (1)
Hidden (2)
Output

Figure 94 : A neural network with 4 layers, which has an input X, with each layer has connecting weights, W_n; sum of a particular node, Z_n; activation of a particular node, A_n; and loss function, L.

Procedure for the calculations to apply a Delta Rule process can be seen in the figure over how to calculate proportional error and turn it into the terms of the delta rule in order to adjust connecting weights, and the figure above is a neural network with 4 layers. Backward calculations are marked as Δ and δ.

After the value for $L(Y, \hat{Y})$ has been obtained, then the Δ for the fourth layer, which is also the Output Layer,

$$\Delta W_3 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial W_3} \qquad \textit{(Eq. 1)}$$

The Δ for Hidden (2) Layer, it is by calculating from the Output Layer

$$\Delta W_2 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2} \qquad \textit{(Eq. 2)}$$

The Δ for Hidden (1) Layer, it is by calculating from the Output and Hidden (2) Layers,

$$\Delta W_1 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial A_1} \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1} \qquad \textit{(Eq. 3)}$$

It can be seen that some variables are repeating and therefore can be factored which is then producing the proportional error δ. Beginning from the Output Layer, partial difference for Loss Function and Activation Function are taken out to form proportional error δ.

$$\delta_3 = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \qquad \textit{(Eq. 4)}$$

Rewriting *Eq. 1* using *Eq. 4*,

$$\Delta W_3 = \delta_3 \cdot \frac{\partial Z_3}{\partial W_3} \qquad \textit{(Eq. 5)}$$

Similarly for Hidden (2) Layer, the δ_3 from *Eq. 4* can also replace variables in *Eq. 2*,

$$\Delta W_2 = \delta_3 \cdot \frac{\partial Z_3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2} \qquad \textit{(Eq. 6)}$$

In *Eq. 6*, the partial difference for Activation over Sum,

$$\frac{\partial Z_3}{\partial A_2} = W_3 \qquad \textit{(Eq. 7)}$$

Then, *Eq. 7* can be replaced into *Eq. 6*

$$\Delta W_2 = \delta_3 \cdot W_3 \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_2} \qquad \textit{(Eq. 8)}$$

From *Eq. 8*, δ for Hidden (2) Layer can be factored,

$$\delta_2 = \delta_3 \cdot W_3 \cdot \frac{\partial A_2}{\partial Z_2} \qquad \textit{(Eq. 9)}$$

Eq. 9 can be replaced into *Eq. 8*

$$\Delta W_2 = \delta_2 \cdot \frac{\partial Z_2}{\partial W_2} \qquad \textit{(Eq. 10)}$$

Similar process for Hidden (2) Layer can be repeated for Hidden (1) Layer, where *Eq. 9* and *Eq. 11* can be replaced into *Eq. 3* to form *Eq. 12*,

$$\frac{\partial Z_2}{\partial A_1} = W_2 \qquad \textit{(Eq. 11)}$$

$$\Delta W_1 = \delta_2 \cdot W_2 \cdot \frac{\partial A_1}{\partial Z_1} \cdot \frac{\partial Z_1}{\partial W_1} \qquad \textit{(Eq. 12)}$$

From *Eq. 12*, δ for Hidden (1) Layer can be factored into,

$$\delta_1 = \delta_2 \cdot W_2 \cdot \frac{\partial A_1}{\partial Z_1} \tag{Eq. 13}$$

Eq. 13 can be rewritten into *Eq. 12*

$$\Delta W_1 = \delta_1 \cdot \frac{\partial Z_1}{\partial W_1} \tag{Eq. 14}$$

Following the calculations from the figure above with 4 layers, the error, δ for layer j can be generalized into

$$\delta_j = \frac{\partial L}{\partial A_j} \cdot \frac{\partial A_j}{\partial Z_j} = \frac{\partial L}{\partial A_j} \sigma'(Z_j) \tag{Eq. 15}$$

and when calculated as a matrix, it must use Hadamard product which is an element-wise product of two vectors,

$$\delta_j = \frac{\partial L}{\partial A_j} \odot \sigma'(Z_j) \tag{Eq. 16}$$

and the δ term can generalized into *Eq. 17* when it is being implemented as a loop function,

$$\delta_j = ((W_{j+1})^T \delta_{j+1}) \odot \sigma'(Z_j) \tag{Eq. 17}$$

For the rate of change of the loss with respect to any bias and any weight in the network,

$$\frac{\partial L}{\partial b_j} = \delta_j \qquad \textit{(Eq. 18)}$$

and

$$\frac{\partial L}{\partial W_j} = \delta_j A_{j-1} \qquad \textit{(Eq. 19)}$$

Updating the weight with the newly calculated gradient and learning rate, α

$$W^+ \leftarrow W + \alpha \frac{\partial L}{\partial W} \qquad \textit{(Eq. 20)}$$

To update the bias with the newly calculated gradient and learning rate, α

$$b^+ \leftarrow b + \alpha \frac{\partial L}{\partial b} \qquad \textit{(Eq. 21)}$$

The backpropagation algorithm with respect to *Mean Squared Error (MSE)* and the output layer is $\hat{Y} = A$, then the gradient for weights and bias can computed:

$$\Delta W = \frac{\partial L}{\partial W} = \frac{\partial L}{\partial A} \cdot \frac{\partial A}{\partial Z} \cdot \frac{\partial Z}{\partial W} = \delta_o \cdot \frac{\partial Z}{\partial W} \qquad \textit{(Eq. 22)}$$

Unpacking the partial derivative,

$$L = \frac{1}{2}\sum(Y-A)^2 \qquad A = \sigma(Z) \qquad Z = W \cdot A_{-1}$$

$$\frac{\partial L}{\partial A} = -(Y-A) \qquad \frac{\partial A}{\partial Z} = \sigma'(Z) \qquad \frac{\partial Z}{\partial W} = A_{-1}$$

$$\frac{\partial L}{\partial W} = \frac{\partial}{\partial W}(\frac{1}{2}\sum(Y-A)^2) \tag*{(Eq. 23)}$$

$$\frac{\partial L}{\partial W} = \frac{1}{2}\sum\frac{\partial}{\partial W}(Y-A)^2 \tag*{(Eq. 24)}$$

$$\frac{\partial L}{\partial W} = \frac{1}{2}\sum\frac{\partial}{\partial A}((Y-A)^2)\cdot\frac{\partial A}{\partial W} \tag*{(Eq. 24)}$$

$$\frac{\partial L}{\partial W} = -(Y-A)\cdot\frac{\partial A}{\partial W} \tag*{(Eq. 25)}$$

$$\frac{\partial L}{\partial W} = -(Y-A)\cdot\frac{\partial A}{\partial Z}\cdot\frac{\partial Z}{\partial W} \tag*{(Eq. 26)}$$

$$\frac{\partial L}{\partial W} = -(Y-A)\cdot\sigma'(Z)\cdot\frac{\partial Z}{\partial W} \tag*{(Eq. 27)}$$

$$\frac{\partial L}{\partial W} = -(Y-A)\cdot\sigma'(Z)\cdot A_{-1} \tag*{(Eq. 28)}$$

$$\delta_o = -(Y-A)\cdot\sigma'(Z) \tag*{(Eq. 29)}$$

$$W = W - \alpha\cdot(Y-A)\cdot\sigma'(Z)\cdot A_{-1} \tag*{(Eq. 30)}$$

Similarly, for bias

$$\Delta b = \frac{\partial L}{\partial b} = \frac{\partial L}{\partial A} \cdot \frac{\partial A}{\partial Z} \cdot \frac{\partial Z}{\partial b} \qquad \textit{(Eq. 31)}$$

$$\frac{\partial Z}{\partial b} = 1 \qquad \textit{(Eq. 32)}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial A} \cdot \frac{\partial A}{\partial Z} \qquad \textit{(Eq. 33)}$$

$$\frac{\partial L}{\partial b} = \delta \qquad \textit{(Eq. 34)}$$

$$\frac{\partial L}{\partial b} = -(Y - A) \cdot \sigma'(Z) \qquad \textit{(Eq. 35)}$$

$$b = b - \alpha \cdot \frac{\partial L}{\partial b} \qquad \textit{(Eq. 36)}$$

For hidden layer,

$$\frac{\partial L}{\partial W_h} = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial W} \qquad \textit{(Eq. 37)}$$

$$\frac{\partial L}{\partial Z_{-1}} = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial Z_{-1}} \qquad \textit{(Eq. 38)}$$

$$\frac{\partial Z}{\partial Z_{-1}} = \frac{\partial Z}{\partial A_{-1}} \cdot \frac{\partial A_{-1}}{\partial Z_{-1}} \qquad \textit{(Eq. 39)}$$

$$\frac{\partial L}{\partial W_{-1}} = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial A_{-1}} \cdot \frac{\partial A_{-1}}{\partial Z_{-1}} \cdot \frac{\partial Z_{-1}}{\partial W_{-1}} \qquad \textit{(Eq. 40)}$$

$$\frac{\partial L}{\partial W_{-1}} = \delta_o \cdot W_{-1} \cdot \sigma'(Z_{-1}) \cdot A_{-1} \qquad \textit{(Eq. 41)}$$

$$\delta_h = \frac{\partial L}{\partial Z} \cdot \frac{\partial Z}{\partial A_{-1}} \cdot \frac{\partial A}{\partial Z_{-1}} = \delta_o \cdot W_{-1} \cdot \sigma'(Z_{-1}) \qquad \textit{(Eq. 42)}$$

Backpropagation for Sigmoid activation is by considering the gradient for the logistic regression which is given by, $X^T(W - Y)$ where Y is a matrix of $N \times 1$.

$$\frac{\partial L}{\partial A} = \frac{\partial}{\partial A}\left[-Y \cdot log\hat{Y} - (1 - Y) \cdot log(1 - \hat{Y})\right] \qquad \textit{(Eq. 44)}$$

$$= -\frac{Y}{\hat{Y}} + \frac{(1 - Y)}{(1 + \hat{Y})} \qquad \textit{(Eq. 45)}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial A} \cdot \frac{\partial A}{\partial Z} \cdot \frac{\partial Z}{\partial W} \qquad \textit{(Eq. 46)}$$

$$= -\left[\frac{Y}{\hat{Y}} + \frac{(1 - Y)}{(1 + \hat{Y})}\right] \cdot \sigma'(Z) \cdot A_{previous} \qquad \textit{(Eq. 47)}$$

$$= -\left[\frac{Y}{\hat{Y}} + \frac{(1 - Y)}{(1 + \hat{Y})}\right] \cdot \hat{Y}(1 - \hat{Y}) \cdot A^2 \qquad \textit{(Eq. 48)}$$

$$= (\hat{Y} - Y) \cdot A_{previous} \qquad \textit{(Eq. 49)}$$

By calculating gradient for Softmax which is similar to the logistic regression, $X^T(W - Y)$ except for Y which is $N \times K$ matrix, one column for each classification.

$$\frac{\partial L}{\partial W} = \sum \frac{\partial L}{\partial A} \cdot \frac{\partial A}{\partial Z} \cdot \frac{\partial Z}{\partial W} \quad \textit{(Eq. 50)}$$

$$= \sum -\frac{Y}{\hat{Y}} \cdot \frac{\partial A}{\partial Z} \cdot A_{previous} \quad \textit{(Eq. 51)}$$

$$\frac{\partial A}{\partial Z} = A \cdot (\delta - \hat{Y}) \quad \textit{(Eq. 52)}$$

Since $A = \hat{Y}$

$$\frac{\partial L}{\partial W} = -\frac{Y}{\hat{Y}} \cdot \hat{Y} \cdot (\delta - \hat{Y}) \cdot A_{previous} \quad \textit{(Eq. 53)}$$

$$= (\hat{Y} - Y) \cdot A_{previous} \quad \textit{(Eq. 53)}$$

Hidden layer

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial A_3} \cdot \frac{\partial A_3}{\partial Z_3} \cdot \frac{\partial Z^3}{\partial A_2} \cdot \frac{\partial A_2}{\partial Z_2} \cdot \frac{\partial Z_2}{\partial W_1} \quad \textit{(Eq. 54)}$$

$$= (\hat{Y} - Y) \cdot W_2 \cdot \sigma'(Z_2) \cdot A_1 \quad \textit{(Eq. 55)}$$

$$= (\hat{Y} - Y) \cdot \sigma'(Z_2) \cdot A_1 \quad \textit{(Eq. 56)}$$

$$= (\hat{Y} - Y) \cdot W_2 \cdot Z_2(1 - Z_2) \cdot A_1 \quad \textit{(Eq. 57)}$$

APPENDIX

Ⓓ

Face Verification, Recognition

and Local Binary Patterns

1.0 FACE VERIFICATION

Face verification is 1×1 comparison and it decides if this two given information matched based on *"Is the person in image B has a similar identity to person in image A?"*, before yielding the output in binary terms of *Yes* or *No*.

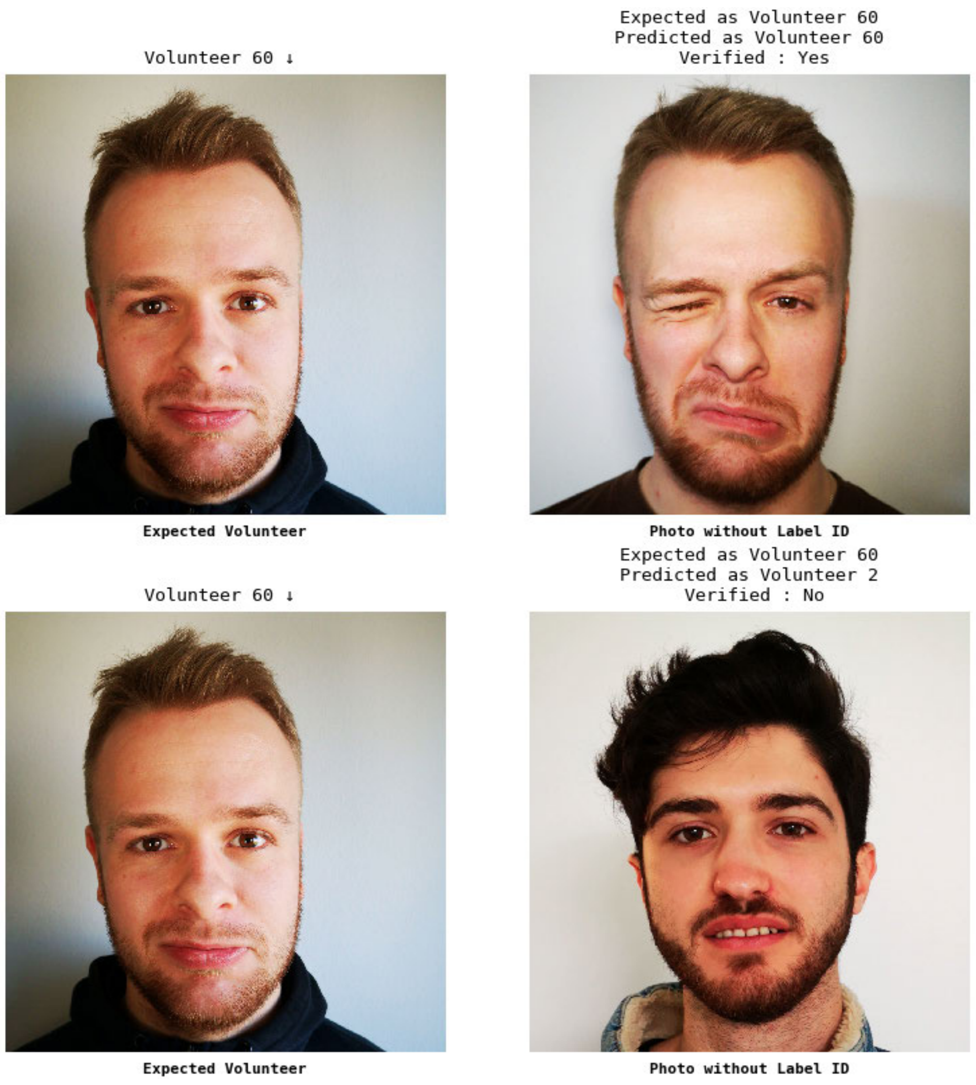


Figure 95 : Verification process is based on known identity and given a new image; the output is expected to answer whether they are of the same person. The first row, the given image is matched to the identity on the left. The second row, the given image is not matched to the identity on the left.

2.0 FACE RECOGNITION

Face recognition is the opposite process from face verification, where face recognition asked *"Whose face is this?"* with an expectation *"This is ... (identity)"* as its answer.

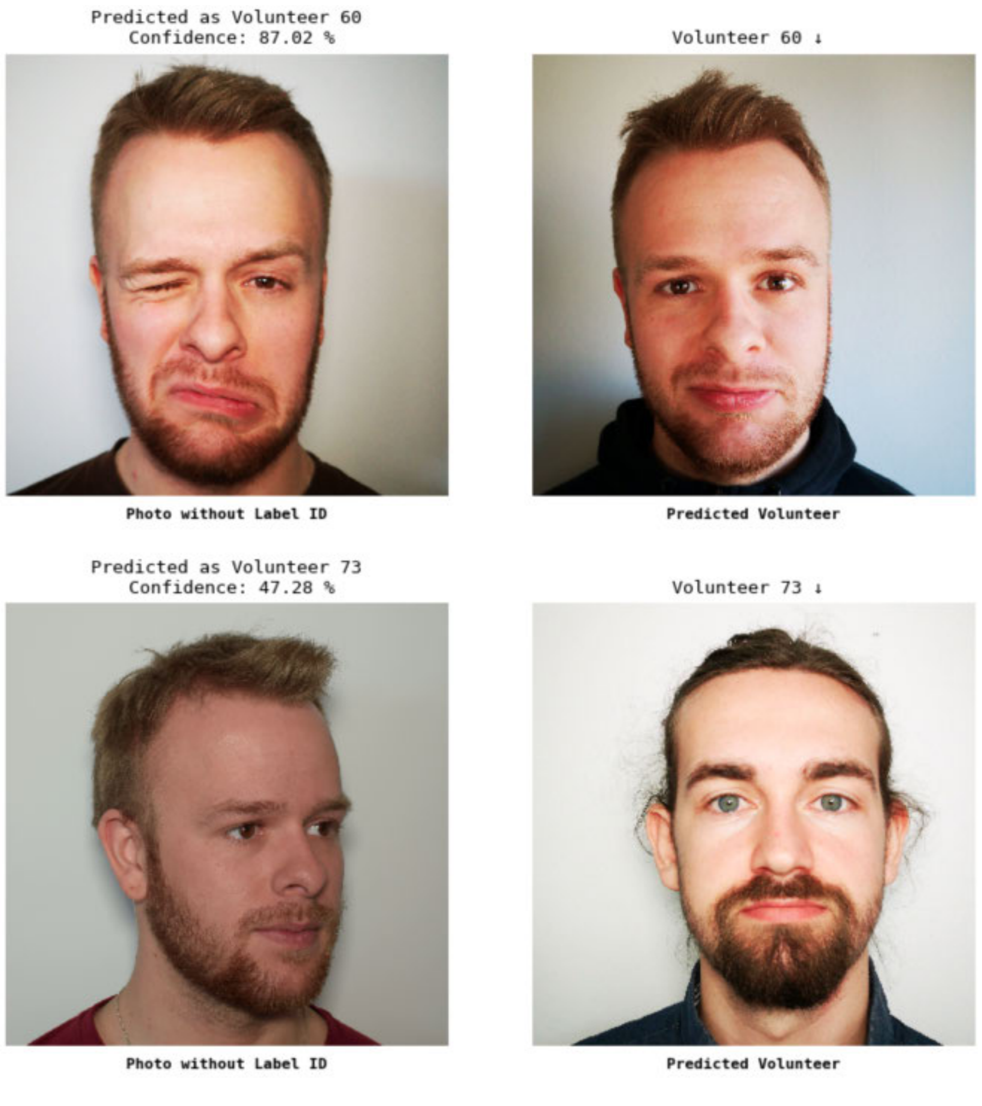


Figure 96 : As opposed to Face Verification, where a base identity was established, in Face Recognition there is no identity base but an image will yield an identity that it has the closest match.

3.0 OPERATIONS FOR LOCAL BINARY PATTERNS (LBP)

LBP Operations as proposed by the original author, will be simplified as follows:

Step 1 : Convert a colour image into a grayscale format.

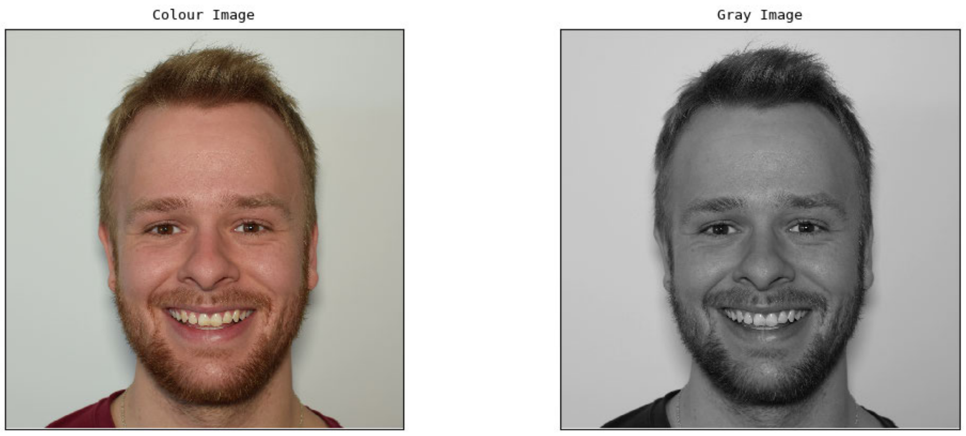


Figure 97 : An original image in colour (left) and converted to a grayscale format (right).

Step 2 : Divide the image into smaller regions, known as grid blocks.

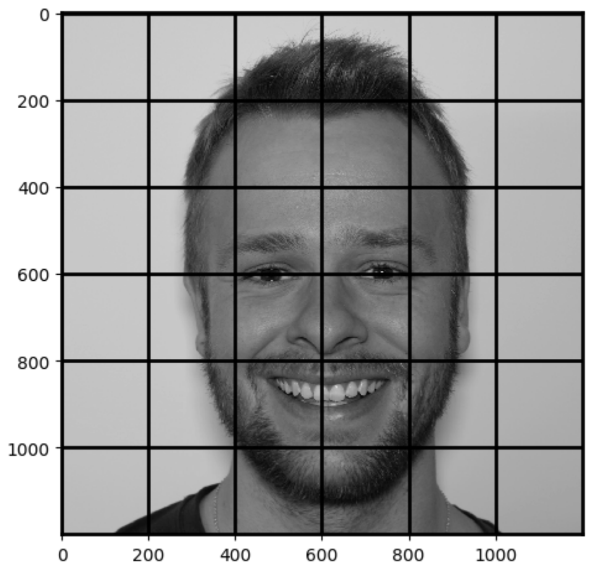


Figure 98 : Grayscale image is divided into grids; in this example it has 64 regions.

Step 3 : For each region divided from Step 2, sub-divide into smaller window for example 3×3. The centre value will be used as a threshold and the other 8 values on the edges are the neighbours.

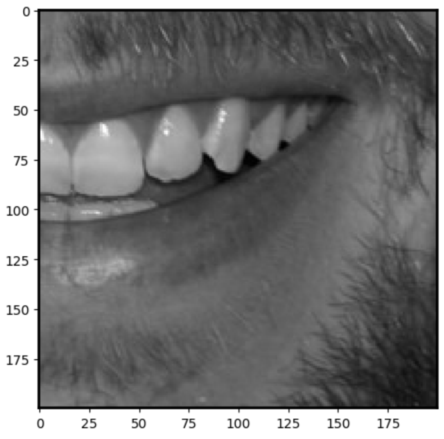


Figure 99 : A region extracted from the image in Step 2 which has 200×200 pixels and when divided into 3×3 neighbourhood window will have 198×198 = 39,204 windows as columns and rows at the edges do not have enough neighbours.

x_4	x_3	x_2
x_5	x	x_1
x_6	x_7	x_8

Figure 100 : A window of 3×3.

Step 4 : Calculate binary values for each neighbour. If $x_i - x \geq 0$ then set value as 1, otherwise set value as 0. Then the number can be concatenated either clockwise or counter clockwise to form a binary number, but the initial place must remain the same when repeating for subsequent windows. As each pixel has 8-digit binary number, it means there are 256 different values possible.

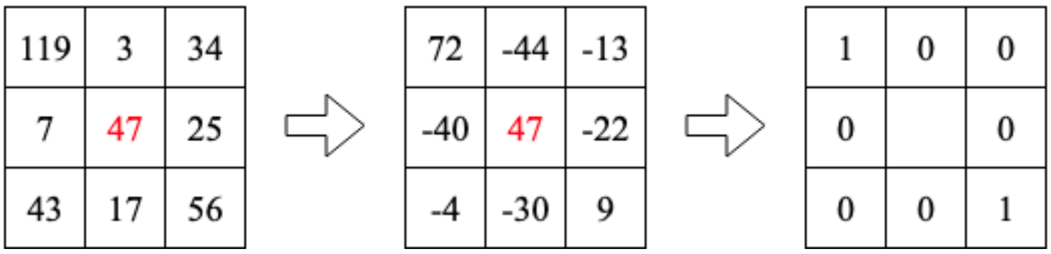
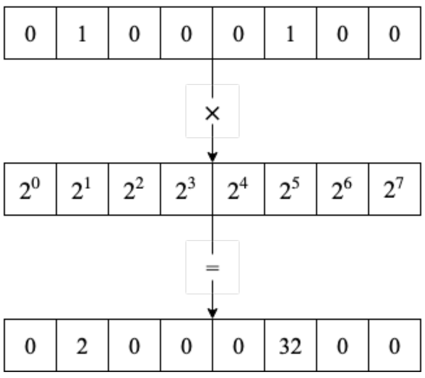


Figure 101 : Operation for thresholding neighbouring values.

Step 5 : Each binary number can be transformed to a decimal and frequencies of these numbers are then counted. The frequencies are then used form a histogram for each grid from *Step 2*. The binary number from the example above is 01000100, and transforming to a decimal,



$= 0 + 2 + 0 + 0 + 0 + 32 + 0 + 0$

$= 34$

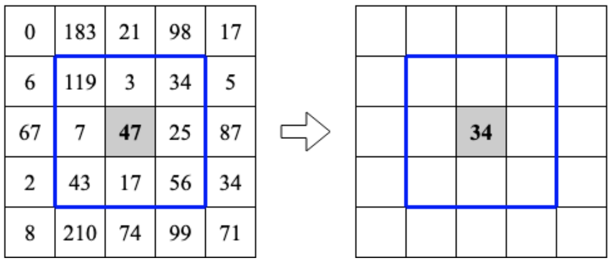


Figure 102 : The calculated LBP is then replaced over the original thresholded value.

Step 6 : Repeat Step 3 through Step 5 to produce LBP image and histogram for each region.

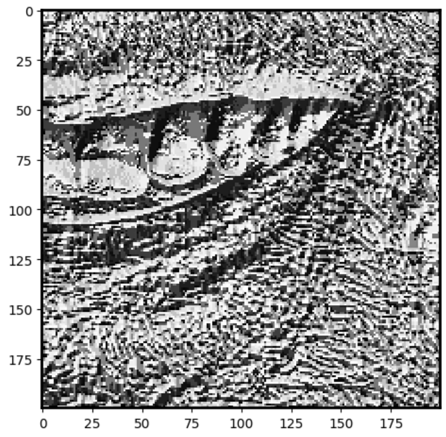


Figure 103 : Following example from Step 3, the image has been turned into an LBP.

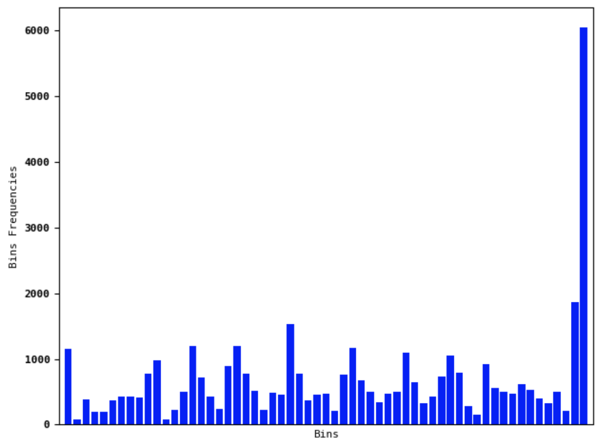


Figure 104 : Histogram result for the particular region.

Step 7 : Concatenate histogram from each grid to form one final characteristic histogram.

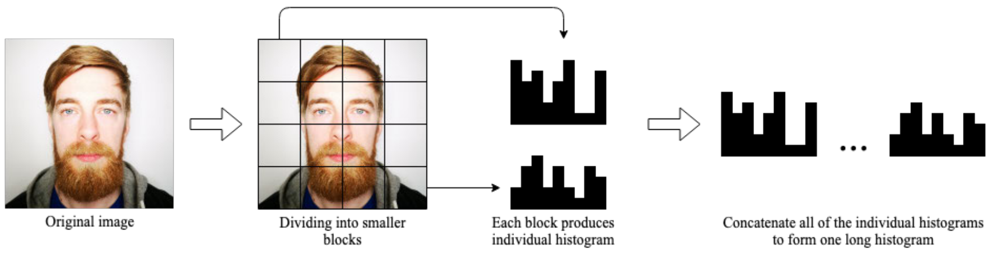

Original image
Dividing into smaller blocks
Each block produces individual histogram
Concatenate all of the individual histograms to form one long histogram

Figure 105 : Visualizing process of transforming an image to a histogram.

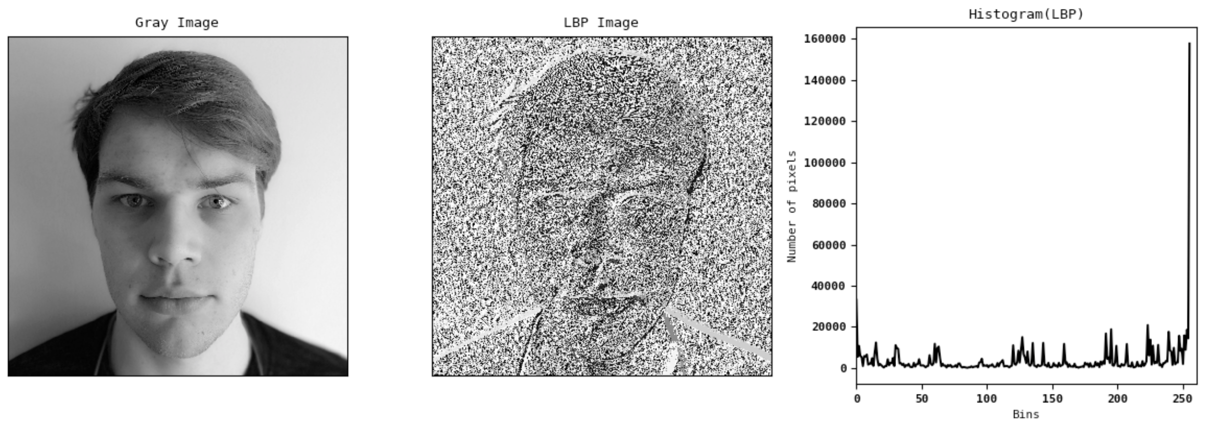


Figure 106 : Grayscale facial image (left) are transformed into Local Binary Patterns (middle); uniformed LBP's occurrences is plotted in the final LBP histogram (right).

The main purpose of using LBP is to reduce a 256-feature vector to only 59 with the main idea behind this is that some binary patterns occur more commonly in textural images, for example the changes on a line which if related to a face can refer to facial expressions. Smile for instance only changes the line the mouth region. Another benefit of using LBPH is that face recognition can be used for images that has been affected by lighting illuminations and can be seen in Figure 107.

The occurrence of unique binary patterns can be observed when a binary number contains at most two of 0 to 1 or 1 to 0 transitions. For example, 0011000 is considered uniform because it has at most two transitions as can be seen at position 2 to 3, where 0 has transitioned into 1 and at position 4 to 5, where 1 has transitioned into 0. The 58 uniform binary patterns in the decimal numbers are: 0, 1, 2, 3, 4, 6, 7, 8, 12, 14, 15, 16, 24, 28, 30, 31, 32, 48, 56, 60, 62, 63, 64, 96, 112, 120, 124, 126, 127, 128, 129, 131, 135, 143, 159, 191, 192, 193, 195, 199, 207, 223, 224, 225, 227, 231, 239, 240, 241, 243, 247, 248, 249, 251, 252, 253, 254 and 255. All other decimal numbers can be grouped into one non-uniform representation at number 59.

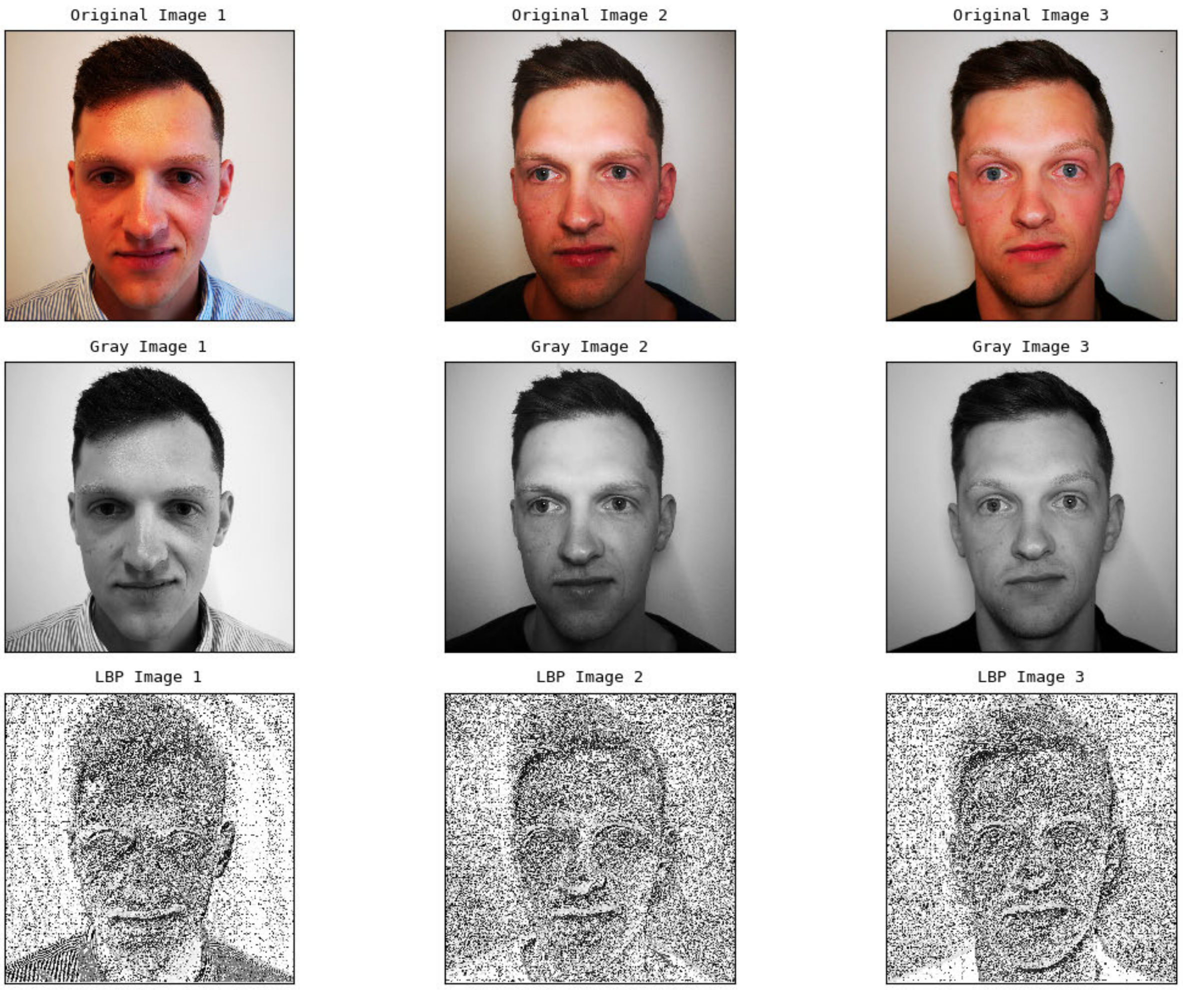


Figure 107 : Three different images with different shades around the eyes, forehead and neck. When transformed to LBP images, the illumination effects are hardly noticeable with facial features look almost similar.

T. Ahonen, A. Hadid, and M. Pietikäinen, “Face description with local binary patterns: application to face recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 12, pp. 2037–2041, Dec. 2006.

APPENDIX

Ⓔ

Reference for Implementation

1.0 MEMORY MANAGEMENT BY USING PYTHON DICTIONARY

Dictionary	Initialization	Key
Parameters	`cache_parameters`	`W` `b`
Calculations	`cache_calculations`	`A` `Z`
Gradients	`cache_gradients`	`dW` `db`
Configuration	`cache_config`	`Learning-Rate` `Mini-Batch`
Analysis	`cache_analysis`	`Loss` `Accuracy`

2.0 VARIABLES SHAPE AND NAME FOR IMPLEMENTATION

Operation	Description	Variable	Shape
Data	Input features	`X`	Matrix
	Target label	`Y` `Y_target`	Matrix
	Association label	`Volunteer_IDs`	$2 \times K$ Matrix
	Binary encoding	`Y_hot` `Y_expected`	Matrix
	Number of features	`n`	Scalar
	Number of samples	`m`	Scalar
	Number of classifications	`K`	Scalar

Operation	**Description**	**Variable**	**Shape**
Configuration	Epoch	`epoch`	Scalar
	Learning rate, alpha	`alpha`	Scalar
	Momentum, mu	`mu`	Scalar
	Mini-batch size	`mini_batch_size`	Scalar
	Lambda regularization	`reg_lambda`	Scalar

Operation	**Description**	**Variable**	**Shape**

Initialization	Weights for particular layer	`W1`
		`W2`
		`W3`
			Matrix

	Bias followed by respective layer number	`b2`
		`b3`
		`b4`
			Matrix

Operation	**Description**	**Variable**	**Type**
Summation and Activation	*Layer 0 - Input* `Z0` = does not exist `A0 = X`	`A0`	Matrix

	Layer 1 - Hidden
	
	`Z1 = W1.T dot A0 + b1`
	`A1 = g(Z1)`
		`Z1`
		`A1`
			Matrix
			Matrix

	Layer 3 - Hidden
	
	`Z2 = W2.T dot A1 + b2`
	`A2 = g(Z2)`
		`Z2`
		`A2`
			Matrix
			Matrix

	Layer 3 - Output
	
	`Z3 = W3.T dot A2 + b3`
	`A3 = g(Z3)`
		`Z3`
		`A3`
			Matrix
			Matrix

Operation	Description	Variable	Shape
Prediction	argmax	`Y_hat`	Matrix
Error / Loss	L(A,Y)	`loss`	Scalar
Backpropagation	Delta	`delta_layer`	Matrix
	dL/dW	`gradient_W`	Matrix
	dL/db	`gradient_b`	Matrix

3.0 DATA SIZE ANALYSIS FOR NEURAL NETWORK

Explanation of data shape by analysing a 4-layer neural network.

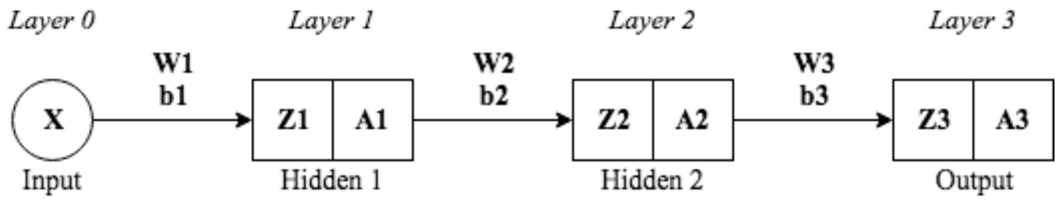


Figure 108 : *Topology of Neural Network for the purpose of analysis.*

	Layer 0	Layer 1	Layer 2	Layer 3
Neurons	12,288	500	100	20

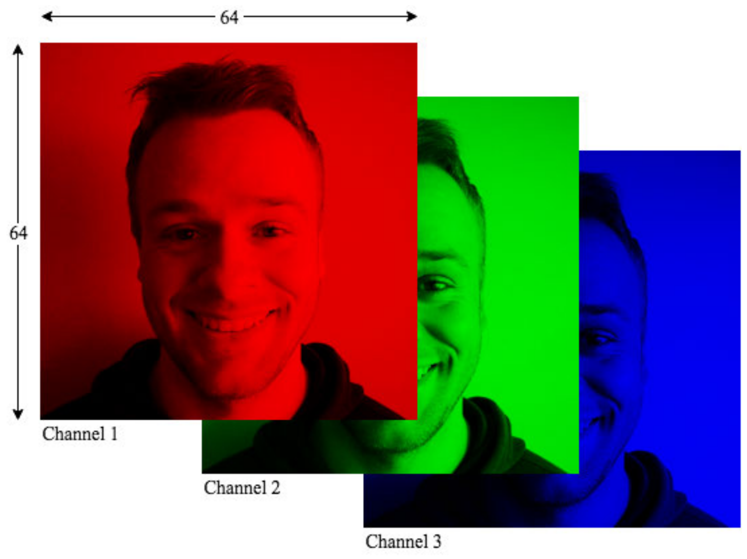


Figure 109 : Visualizing an image with 64×64 together with its colour channels. Number of features = height × width × depth = 64 × 64 × 3 = 12,288.

Variable	**Shape**
`X`	12,288 × 200
`Y_expected`	1 × 200
`Volunteer_IDs`	1 × 200
`Y_hot`	20 × 200
`n`	12,288
`m`	200
`K`	20

Variable	**Shape**

`W1`
`W2`
`W3`
	12,288 × 500
	500 × 100
	100 × 20

`b1`
`b2`
`b3`
	500 × 1
	100 × 1
	20 × 1

Operation : Initialization

Variable	**Shape**
Layer 0 `A0 = X`	 `12,288 × 200`

Layer 1
`Z1 = [500 × 12,288] • [12,288 × 200]`
`A1 = activate(Z1)`
	
	`500 × 200`
	`500 × 200`

Layer 2
`Z2 = [100 × 500] • [500 × 200]`
`A2 = activate(Z2)`
	
	`100 × 200`
	`100 × 200`

Layer 3
`Z3 = [20 × 100] • [100 × 200]`
`A3 = activate(Z3)`
	
	`20 × 200`
	`20 × 200`

Operation : Summation and Activation

APPENDIX

Ⓕ

Dataset for Training and Testing

TRAINING DATASET

Training for the neural network is based on a set of facial images taken from with different facial expressions as well as different movement of the face. The purpose of this is so that a neural network can *learn* and make a generalization from one volunteer from a different angle and different muscle movement. The argument for this is that face recognition supposed to be able to make recognition not only from a forward looking and neutral expression. Face recognition should consider the environment in terms of lighting and illumination, natural facial expressions as well as non-natural facial expressions. This is because, a person might be having a certain emotion during which time he is being photograph. This will expand the possibility to make face recognition better.

Forward	Looking at camera
Semi-sideway	Between 30 to 60 degrees (both sides)
Sideway	Between 60 to 90 degrees
Upward	Between 30 to 60 degrees
Downward	Between 30 to 60 degrees
Mixed	Semi-sideway and facing upward Semi-sideway and facing downward

Table 21 : Volunteers will be asked to move his/her face in different angles.

Examples of 100 photos from one volunteer showing different facial expressions in different angles can be seen in the next 5 pages.

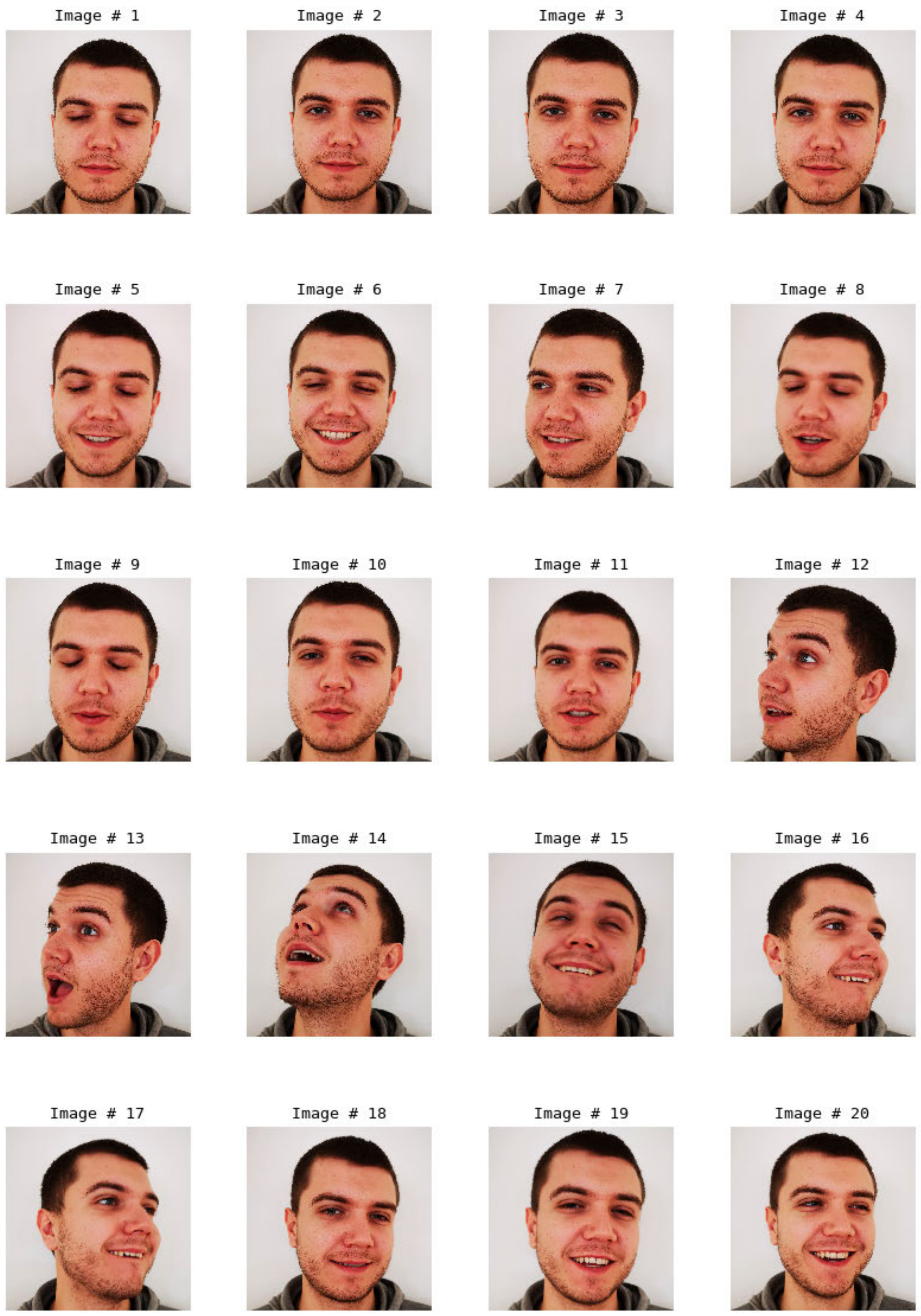


Image # 1
Image # 2
Image # 3
Image # 4
Image # 5
Image # 6
Image # 7
Image # 8
Image # 9
Image # 10
Image # 11
Image # 12
Image # 13
Image # 14
Image # 15
Image # 16
Image # 17
Image # 18
Image # 19
Image # 20

Image # 21

Image # 22

Image # 23

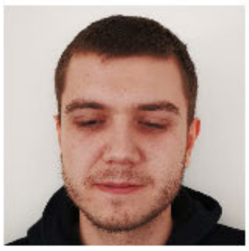


Image # 24

Image # 25

Image # 26

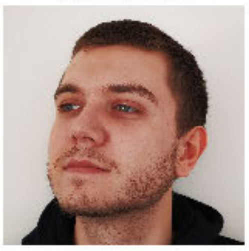


Image # 27

Image # 28

Image # 29

Image # 30

Image # 31

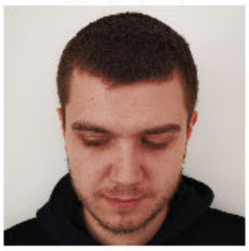


Image # 32

Image # 33

Image # 34

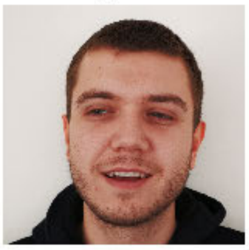


Image # 35

Image # 36

Image # 37

Image # 38

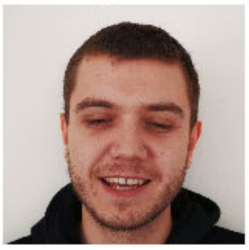


Image # 39

Image # 40

Image # 41

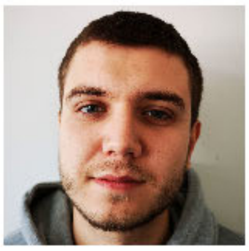



Image # 42

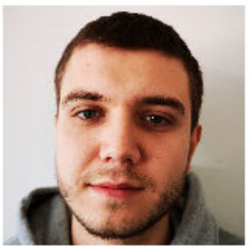



Image # 43

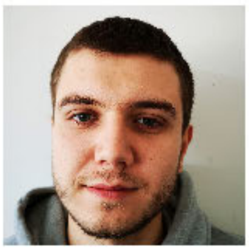



Image # 44

Image # 45

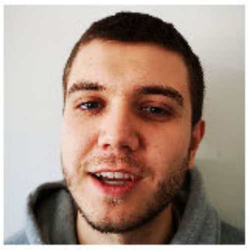



Image # 46

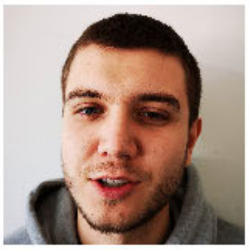



Image # 47

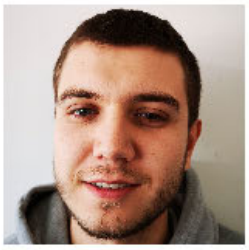



Image # 48

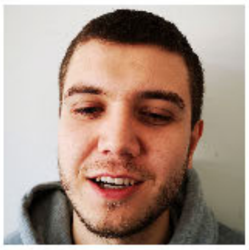



Image # 49

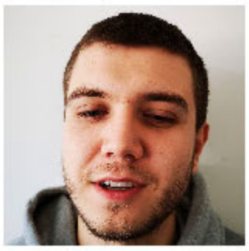



Image # 50

Image # 51

Image # 52

Image # 53

Image # 54

Image # 55

Image # 56

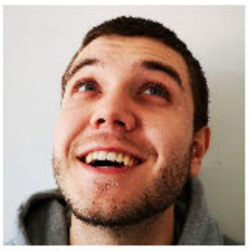



Image # 57

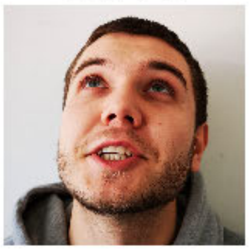



Image # 58

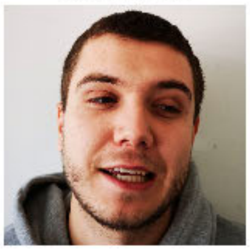



Image # 59

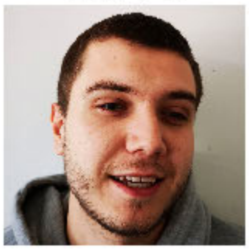



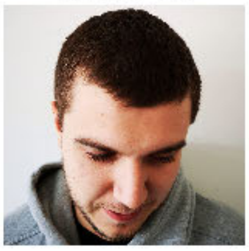
Image # 60


Image # 61

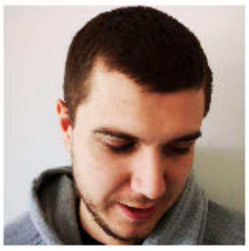



Image # 62

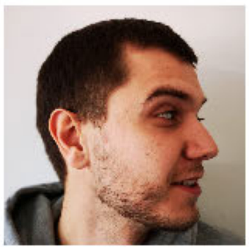



Image # 63

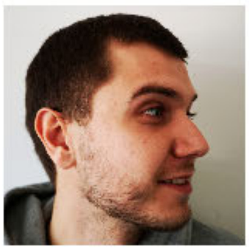



Image # 64

Image # 65

Image # 66

Image # 67

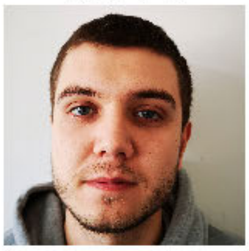



Image # 68

Image # 69

Image # 70

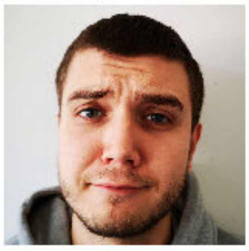



Image # 71

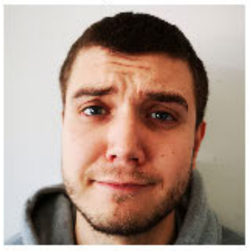



Image # 72

Image # 73

Image # 74

Image # 75

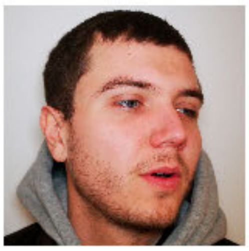



Image # 76

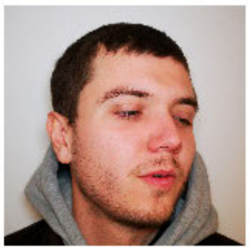



Image # 77

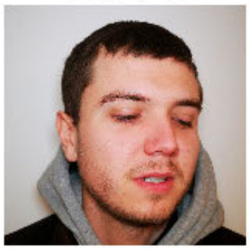



Image # 78

Image # 79

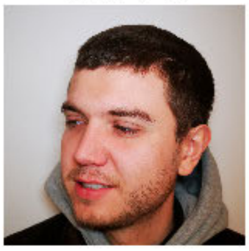



Image # 80

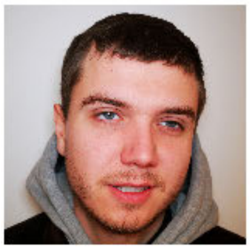


Image # 81

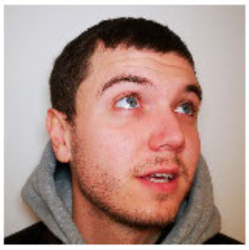


Image # 82

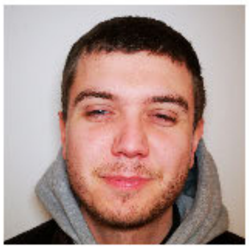


Image # 83

Image # 84

Image # 85

Image # 86

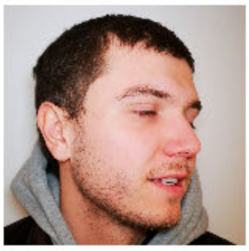


Image # 87

Image # 88

Image # 89

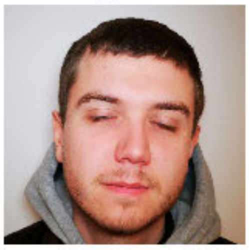


Image # 90

Image # 91

Image # 92

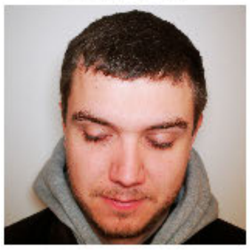


Image # 93

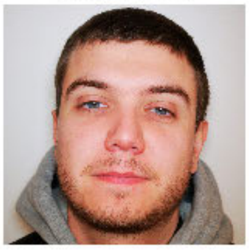


Image # 94

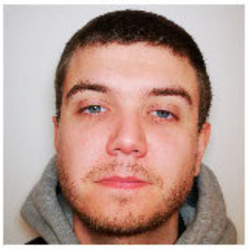


Image # 95

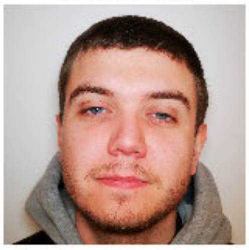


Image # 96

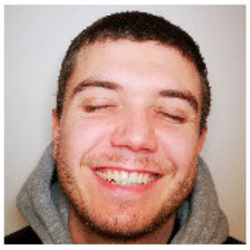


Image # 97

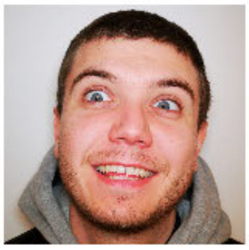


Image # 98

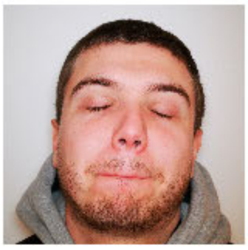


Image # 99

Image # 100